(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 108234302 A
(43)申请公布日 2018.06.29

(21)申请号 201711230685.9

(22)申请日 2017.11.29

(30)优先权数据
62/437,369 2016.12.21 US
62/479,804 2017.03.31 US
15/637,839 2017.06.29 US

(71)申请人 丛林网络公司
地址 美国加利福尼亚

(72)发明人 戴维·M·卡茨 罗斯·W·卡隆
斯科特·麦凯
丹尼斯·C·弗格森

(74)专利代理机构 北京康信知识产权代理有限责任公司 11240
代理人 梁丽超 田喜庆

(51)Int.Cl.
H04L 12/703(2013.01)
G06F 9/50(2006.01)
G06F 11/14(2006.01)

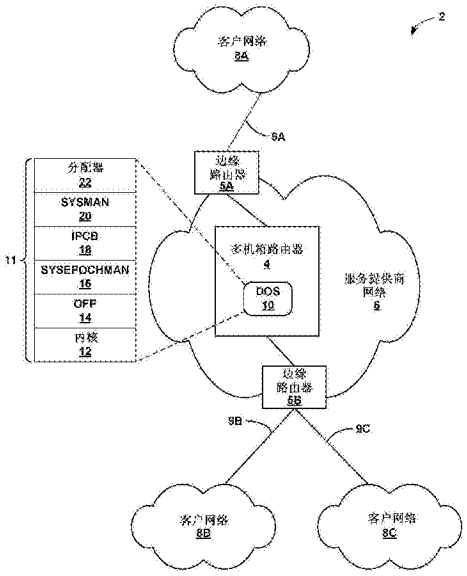
权利要求书3页 说明书40页 附图13页

(54)发明名称

保持网络装置用的分布式操作系统中的一致性

(57)摘要

本发明涉及保持网络装置用的分布式操作系统中的一致性。总体上，描述用于保持网络装置用的分布式操作系统中的一致性的技术。包括硬件计算节点的网络装置可以被配置为执行该技术。硬件计算节点可执行分布式操作系统。至少一个硬件计算节点可确定多个硬件计算节点中的一个或多个硬件计算节点是否已出现故障并且不再支持分布式操作系统的执行，并且确定多个硬件计算节点的剩余硬件计算节点是否超过群体阈值。硬件计算节点中的至少一个可在多个硬件计算节点的剩余硬件计算节点小于群体阈值时，进一步重新启动分布式操作系统。



CN 108234302 A

1.一种方法,包括:

由包括在网络装置内的多个硬件计算节点中的至少一个硬件计算节点确定所述多个硬件计算节点中的一个或多个硬件计算节点是否发生了故障;

由所述多个硬件计算节点中的所述至少一个硬件计算节点确定所述多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值;以及

当所述多个硬件计算节点中的所述剩余硬件计算节点小于所述群体阈值时,由所述多个硬件计算节点中的所述至少一个硬件计算节点重新启动分布式操作系统。

2.根据权利要求1所述的方法,进一步包括:

设定纪元值以表示所述分布式操作系统的当前版本;以及

将所述纪元值增加1,使得所述分布式操作系统的实例避免执行所述分布式操作系统的两个不同版本。

3.根据权利要求1或2所述的方法,进一步包括:

标识所述多个硬件计算节点中的被配置为会聚于所述分布式操作系统的单个版本的群体;

将所述多个硬件计算节点的所述群体中的一个硬件计算节点选择为纪元管理器主控硬件计算节点;以及

由所述纪元管理器主控硬件计算节点确定所述多个硬件计算节点的所述群体中的连接是否已发生故障。

4.根据权利要求3所述的方法,其中,所述群体阈值等于所述多个硬件计算节点的所述群体中的数量除以2加1:(N/2)+1,其中,N为所述多个硬件计算节点的所述群体中的数量。

5.根据权利要求3所述的方法,进一步包括:

由参与所述群体的所述多个硬件计算节点中的一个硬件计算节点确定参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点丧失了到所述群体的连接;以及

响应于所述多个硬件计算节点中的所述一个硬件计算节点丧失了到所述群体的连接的确定,由参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点重新启动参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点。

6.根据权利要求3所述的方法,进一步包括:在连接故障不超过连接故障阈值时,执行当前版本的所述分布式操作系统。

7.根据权利要求1或2所述的方法,进一步包括:

执行协议,通过所述协议发现所述多个硬件计算节点的拓扑;

基于所述拓扑确定所述多个硬件计算节点的子集,以管理所述分布式操作系统的执行;

执行通信总线,通过所述通信总线在所述多个硬件计算节点的子集内同步操作系统状态信息;以及

基于所述操作系统状态信息执行所述分布式操作系统,以提供一个或多个应用执行的执行环境。

8.一种网络装置,包括:

多个硬件计算节点,被配置为执行分布式操作系统,所述多个硬件计算节点中的至少一个硬件计算节点被配置为:

确定所述多个硬件计算节点中的一个或多个硬件计算节点是否发生了故障并且不再支持所述分布式操作系统的执行；

确定所述多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值；以及

当所述多个硬件计算节点中的所述剩余硬件计算节点小于所述群体阈值时，重新启动所述分布式操作系统。

9.根据权利要求8所述的网络装置，其中，所述分布式操作系统进一步被配置为：

设定纪元值以表示所述分布式操作系统的当前版本；以及

将所述纪元值增加1，使得所述分布式操作系统的实例避免执行所述分布式操作系统的两个不同版本。

10.根据权利要求8或9所述的网络装置，其中，所述分布式操作系统进一步被配置为：

标识所述多个硬件计算节点中的被配置为会聚于所述分布式操作系统的单个版本的群体；以及

将所述多个硬件计算节点的所述群体中的一个硬件计算节点选择为纪元管理器主控硬件计算节点，

其中，所述纪元管理器主控硬件计算节点确定所述多个硬件计算节点的所述群体中的连接是否已发生故障。

11.根据权利要求10所述的网络装置，其中，所述群体阈值等于所述多个硬件计算节点的所述群体中的数量除以2加1：(N/2)+1，其中，N为所述多个硬件计算节点的所述群体中的数量。

12.根据权利要求10所述的网络装置，

其中，参与所述群体的所述多个硬件计算节点中的一个硬件计算节点被配置为：

确定参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点丧失了到所述群体的连接，以及

响应于所述多个硬件计算节点中的所述一个硬件计算节点丧失了到所述群体的连接的确定，重新启动参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点。

13.根据权利要求10所述的网络装置，其中，所述多个硬件计算节点被配置为在连接故障不超过连接故障阈值时，执行当前版本的所述分布式操作系统。

14.根据权利要求8或9所述的网络装置，其中，所述多个硬件计算节点进一步被配置为：

执行协议，通过所述协议发现所述多个硬件计算节点的拓扑；

基于所述拓扑确定所述多个硬件计算节点的子集，以管理所述分布式操作系统的执行；

执行通信总线，通过所述通信总线在所述多个硬件计算节点的子集内同步操作系统状态信息；

基于所述操作系统状态信息执行所述分布式操作系统，以提供一个或多个应用执行的执行环境。

15.一种非暂时性计算机可读存储介质，其上存储有指令，所述指令在被执行时使网络装置的一个或多个处理器被配置为：

确定执行分布式操作系统的多个硬件计算节点中的一个或多个硬件计算节点是否发

生了故障；

确定所述多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值；以及

当所述多个硬件计算节点中的所述剩余硬件计算节点小于所述群体阈值时，重新启动所述分布式操作系统。

16.根据权利要求15所述的非暂时性计算机可读存储介质，其上进一步存储有指令，所述指令在被执行时使所述一个或多个处理器被配置为：

设定纪元值以表示所述分布式操作系统的当前版本；以及

将所述纪元值增加1，使得所述分布式操作系统的实例避免执行所述分布式操作系统的两个不同版本。

17.根据权利要求15或16所述的非暂时性计算机可读存储介质，其上进一步存储有指令，所述指令在被执行时使所述一个或多个处理器被配置为：

标识所述多个硬件计算节点中的被配置为会聚于所述分布式操作系统的单个版本的群体；

将所述多个硬件计算节点的所述群体中的一个硬件计算节点选择为纪元管理器主控硬件计算节点；以及

由所述纪元管理器主控硬件计算节点确定所述多个硬件计算节点的所述群体中的连接是否已发生故障。

18.根据权利要求17所述的非暂时性计算机可读存储介质，其中，所述群体阈值等于所述多个硬件计算节点的所述群体中的数量除以2加1：(N/2)+1，其中，N为所述多个硬件计算节点的所述群体中的数量。

19.根据权利要求17所述的非暂时性计算机可读存储介质，其上进一步存储有指令，所述指令在被执行时使所述一个或多个处理器被配置为：

由参与所述群体的所述多个硬件计算节点中的一个硬件计算节点确定参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点丧失了到所述群体的连接；以及

响应于所述多个硬件计算节点中的所述一个硬件计算节点丧失了到所述群体的连接的确定，由参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点重新启动参与所述群体的所述多个硬件计算节点中的所述一个硬件计算节点。

20.根据权利要求17所述的非暂时性计算机可读存储介质，其上进一步存储有指令，所述指令在被执行时使所述一个或多个处理器被配置为在连接故障不超过连接故障阈值时，执行当前版本的所述分布式操作系统。

保持网络装置用的分布式操作系统中的一致性

[0001] 本申请要求于2016年12月21日提交的美国临时申请第62/437,369号和于2017年3月31日提交的美国临时申请第62/479,804号的权益,其全部内容均通过引用结合于此。

技术领域

[0002] 本公开涉及网络装置,并且更具体地,涉及用于网络装置的操作系统。

背景技术

[0003] 用于诸如路由器的网络装置的操作系统用于提供其中各种应用(诸如,网络协议、维护接口、虚拟化应用等)可以执行的执行环境。路由器的操作系统的一个功能是建立一种方式,通过该方式可以将状态信息传播到各种组件(或换言之,路由器的计算节点),以允许路由器在网络内正确地路由网络流量。

[0004] 例如,路由器可以维护表示路由器与网络之间的接口的当前状态的状态信息。这种状态信息可以包括表示一个或多个IFC的状态的信息,例如,IFC的当前配置。作为另外的示例,路由器可以维护表示路由器内的一个或多个分组转发引擎(PFE)、一个或多个路由引擎或其他资源的状态的状态信息。

[0005] 在路由器内操作的被称为“路由引擎”的控制节点可以执行操作系统的实例,以将状态信息(及其变化)传播到路由器内的各种其他进程或组件。这些其他进程或组件有时被称为“消费者”,因为其接收和利用(换言之,“消费”)由操作系统的实例维护的状态信息。这些消费者在执行各种功能时利用状态信息。

[0006] 由于近年来传统网络的复杂性增加,所以路由器或其他网络装置内的状态信息的管理同样成为重大挑战。一些现有的用于管理状态信息的方法涉及在操作系统的实例内缓存信息,并向在路由器内执行的消费者发出状态更新通知消息。作为响应,消费者从操作系统的实例中检索状态信息。

[0007] 为了增加可靠性,一些路由器可以包括主要路由引擎和一个或多个备用路由引擎,每个引擎可以执行操作系统的单独的不同实例,来管理状态信息。在主要路由引擎发生故障的情况下,一个备用路由引擎承担对路由资源的控制,以继续路由器的操作。在主要路由引擎和备用路由引擎之间切换路由功能的控制处理通常被称为故障切换。在一些情况下,为了采取适当的控制并确保操作,备用路由引擎被迫例如通过将路由器资源电源循环到已知状态来从每个资源“重新学习”丢失的状态信息。由于备用路由引擎执行的操作系统实例重建正确的状态信息,因此会在路由器资源重新启动操作时导致分组转发中断。

[0008] 路由器不仅发展得更加可靠,而且满足日益增长的带宽需求。满足日益增长的带宽需求的一种方法是使用多机箱路由器,即其中多个路由装置在物理上耦接并且被配置为作为单个路由器操作的路由器。例如,多机箱路由器可以包括多个线卡机箱(LCC),该线卡机箱包括一个或多个IFC以及在LCC之间转发分组并且提供多机箱路由器的自上向下管理的中央交换卡机箱(SCC)。由于多机箱路由器组合多个路由装置的资源,所以多机箱路由器通常具有比独立路由器高得多的带宽能力。通过将路由功能整合到更少的路由器上,使用

多机箱路由器可以简化和改善服务提供商网络上的路由。

[0009] 然而，多机箱路由器可能导致大量不同的组件(例如，路由引擎)，每个组件执行需要正确维护状态信息并将状态信息的变化传送给下游消费者的操作系统的不同实例。即，除了均执行操作系统的不同实例的多个路由引擎之外，多机箱路由器还可以包括SCC和LCC，SCC和LCC还包括执行操作系统的另一实例的控制节点，所有这些都需要状态信息的至少一部分并且将状态信息传播给一些(如果不是全部)各种消费者。

发明内容

[0010] 描述了用于为网络装置提供分布式操作系统的技术，该技术可以允许底层硬件资源的动态扩张或收缩(换言之，“弹性”)，同时还可能在生产组件(所谓的“生产商”)和消费组件(所谓的“消费者”)上提供状态信息的稳健会聚。操作系统可以分布在计算节点(其也可以被称为“硬件计算节点”、“计算节点”或“节点”)上，计算节点可以包括路由引擎、接口卡、分组转发引擎以及非网络节点，诸如处理器、中央处理单元(CPU)、专用集成电路(ASIC)、图形处理单元(GPU)。计算节点最初可以从内核开始合并，经由以类似于链路状态路由协议的拓扑发现为前提的对象洪泛协议(OFP)来检测彼此，并且经由管理进程(被称为“SysEpochMan”)进行组织，以执行分布式操作系统。

[0011] 一旦在计算节点上启动，分布式操作系统可以允许实时(或接近实时地)构建用于同步由分布式操作系统维护的状态信息的数据库的同步树。操作系统可以使用OFP同步数据库，同时还可能修剪数据库并减少带宽需求。操作系统可以使用被称为“系统纪元(system epoch)”的系统进程处理在执行分布式操作系统的实例的计算节点之间的一致性(coherence)，使得在分布式操作系统的实例发生的各种连接或故障的情况下，可以由每个实例维护一致的状态信息。

[0012] 分布式操作系统可以避免相同操作系统的单独实例的冗余执行，同时通过以状态信息的多播传送形式的洪泛来简化状态信息的传播。此外，分布式操作系统可能对计算节点故障具有弹性，从而允许支持分布式操作系统的单独计算节点发生故障，而不需要重新启动支持分布式操作系统的单个实例的执行的剩余计算节点。

[0013] 执行分布式操作系统的相应实例的网络装置的计算节点可以被配置为将状态信息存储在相应数据结构(例如，树数据结构)中。网络装置的计算节点可以将状态信息表示为消息片段，其中，每个消息片段存储在树数据结构的树节点中。此外，网络装置的计算节点可以通过将表示消息片段的对象洪泛到其他计算节点，来使用OFP同步树数据结构。例如，当网络装置的一个计算节点接收到更新的状态信息时，网络装置的该一个计算节点可以更新其本地树数据结构，然后，根据OFP将更新的状态信息洪泛到网络装置的其他计算节点。以这种方式，计算节点可以在其相应的用于存储分布式操作系统的和/或在由分布式操作系统提供的应用空间中执行的应用的状态信息的数据结构之间，保持同步。

[0014] 在一个示例中，一种方法包括：由网络装置的在电路中实现的执行分布式操作系统的第一实例的第一计算节点接收用于分布式操作系统和在由分布式操作系统提供的应用空间中执行的应用中的至少一者的更新的状态信息。该方法还包括由网络装置的第一计算节点更新网络装置的第一计算节点的本地数据结构，以包括更新的状态信息，本地数据结构存储多个对象，每个对象定义用于分布式操作系统和应用中的至少一者的状态信息的

一部分。该方法还包括由网络装置的第一计算节点使更新的本地数据结构，与由网络装置的在电路中实现的第二节点执行的分布式操作系统的第二实例的远程数据结构同步。

[0015] 在另一示例中，一种网络装置包括：在电路中实现的第一硬件节点；以及在电路中实现的第二硬件节点。第一硬件节点被配置为：执行分布式操作系统的第一实例；并且维护存储多个对象的第一数据结构，每个对象定义分布式操作系统和在由分布式操作系统提供的应用空间中执行的应用中的至少一者的状态信息的一部分。第二硬件节点被配置为：执行分布式操作系统的第二实例；并且维护存储多个对象的同步版本的第二数据结构。第一硬件节点被进一步配置为：接收用于分布式操作系统和应用中的至少一者的更新的状态信息；更新第一数据结构，以包括更新的状态信息；并且通过执行分布式操作系统的第一实例来使更新的第一数据结构与第二数据结构同步。第二硬件节点被进一步配置为：通过执行分布式操作系统的第二实例，使第二数据结构与更新的第一数据结构同步。

[0016] 在另一示例中，一种其上存储有指令的非暂时性计算机可读存储介质，该指令在执行时使网络装置的第一计算节点的第一处理器：执行分布式操作系统的第一实例，以接收用于分布式操作系统和在由分布式操作系统提供的应用空间中执行的应用中的至少一者的更新的状态信息；更新网络装置的第一计算节点的本地数据结构，以包括更新的状态信息，本地数据结构存储多个对象，每个对象定义用于分布式操作系统和应用中的至少一者的状态信息的一部分；以及使更新的本地数据结构与由网络装置的第二计算节点执行的分布式操作系统的第二实例的远程数据结构同步。

[0017] 在另一示例中，一种装置包括多个硬件计算节点，硬件计算节点被配置为执行协议，通过该协议发现多个硬件计算节点的拓扑，并且基于拓扑确定多个硬件计算节点的子集，以管理分布式操作系统的执行。多个硬件计算节点的所确定的子集被进一步配置为执行通信总线，通过通信总线来同步多个硬件计算节点的子集之间的操作系统状态信息。多个硬件计算节点被进一步配置为基于操作系统状态信息来执行分布式操作系统，以提供其中一个或多个应用执行的执行环境。

[0018] 在另一示例中，一种方法包括由多个硬件计算节点执行协议，通过该协议来发现多个硬件计算节点的拓扑，并且由多个硬件计算节点中的至少一个并且基于拓扑确定多个硬件计算节点的子集，以管理分布式操作系统的执行。该方法还包括由多个硬件计算节点的所确定的子集执行通信总线，通过通信总线来同步多个硬件计算节点的子集之间的操作系统状态信息，并且由多个硬件计算节点并且基于操作系统状态信息执行分布式操作系统，以提供其中一个或多个应用执行的执行环境。

[0019] 在另一示例中，一种其上存储有指令的非暂时性计算机可读存储介质，该指令在执行时使：多个硬件计算节点中的一个或多个执行协议，通过该协议来发现多个硬件计算节点的拓扑；基于拓扑确定多个硬件计算节点的子集，以管理分布式操作系统的执行；执行通信总线，通过通信总线来同步多个硬件计算节点的子集之间的操作系统状态信息；以及基于操作系统状态信息执行分布式操作系统，以提供其中一个或多个应用执行的执行环境。

[0020] 在另一示例中，网络装置包括多个硬件计算节点，硬件计算节点被配置为执行分布式操作系统，多个硬件计算节点中的至少一个被配置为确定多个硬件计算节点中的一个或多个是否发生故障并且不再支持分布式操作系统的执行。多个硬件计算节点中的至少一

个被进一步配置为确定多个硬件计算节点中的剩余硬件计算节点是否超过群体(quorum)阈值,并且当多个硬件计算节点中的剩余硬件计算节点小于群体阈值时,重新启动分布式操作系统。

[0021] 在另一示例中,一种方法包括:由包括在网络装置内的多个硬件计算节点中的至少一个确定多个硬件计算节点中的一个或多个是否发生故障;由多个硬件计算节点中的至少一个确定多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值;并且当多个硬件计算节点中的剩余硬件计算节点小于群体阈值时,由多个硬件计算节点中的至少一个重新启动分布式操作系统。

[0022] 在另一示例中,一种其上存储有指令的非暂时性计算机可读存储介质,该指令在执行时使网络装置的一个或多个处理器:确定执行分布式操作系统的多个硬件计算节点中的一个或多个是否发生故障;确定多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值;并且当多个硬件计算节点中的剩余硬件计算节点小于群体阈值时,重新启动分布式操作系统。

[0023] 在附图和下面的描述中阐述了这些技术的一个或多个方面的细节。这些技术的其他特征、目的和优点将从描述和附图以及权利要求中显而易见。

附图说明

[0024] 图1是示例性网络计算环境的框图,在该网络计算环境中,服务提供商网络包括被配置为根据本公开中描述的分布式操作系统技术进行操作的多机箱路由器。

[0025] 图2是示出被配置为根据本公开中描述的分布式操作系统技术进行操作的示例性多机箱路由器的框图。

[0026] 图3是示出被配置为根据本公开中描述的分布式操作系统技术的各种方面进行操作的图2所示的多机箱路由器的示例性节点的框图。

[0027] 图4A至图8B是示出在解决可能影响根据本公开中描述的技术维护的分布式操作系统的执行的各种情况时图2所示的多机箱路由器内的节点操作的各种方面的框图。

[0028] 图9是示出图3所示的多机箱路由器的节点在执行本公开中描述的分布式操作系统技术的各种方面时的示例性操作的流程图。

[0029] 图10是示出根据本公开的技术的用于存储状态信息的示例树数据结构的概念图。

[0030] 图11是示出根据本公开的技术的用于在由网络装置的相应计算节点执行的分布式操作系统的不同实例之间同步状态信息的示例性方法的流程图。

具体实施方式

[0031] 图1是示出其中服务提供商网络6包括多机箱路由器4的示例性计算环境2的框图。在该示例中,多机箱路由器4与边缘路由器5A和5B(“边缘路由器5”)通信,以向客户网络8A-8C(“客户网络8”)提供对网络6的访问。在一个实施方式中,多机箱路由器4包括作为控制节点操作的中央交换卡机箱(SCC)和作为分组路由装置操作的一个或多个线卡机箱(LCC)。LCC可以包含用于耦接到网络6内的其他装置的所有物理接口,而SCC控制并且路由LCC之间的流量。

[0032] 尽管未示出,但是服务提供商网络6可以耦接到由其他提供商管理的一个或多个

网络，并且因此可以形成大规模公共网络基础设施(例如，互联网)的一部分。因此，客户网络8可以被视为互联网的边缘网络。服务提供商网络6可以向客户网络8内的计算装置提供对互联网的访问，并且可以允许客户网络8内的计算装置彼此通信。服务提供商网络6可以包括除了多机箱路由器4和边缘路由器5之外的各种网络装置，诸如，附加的路由器、交换机、服务器或其他装置。

[0033] 在所示实施方式中，边缘路由器5A经由接入链路9A耦接到客户网络8A，并且边缘路由器5B经由接入链路9B和9C分别耦接到客户网络8B和8C。客户网络8可以是用于企业的地理上分离的站点的网络。客户网络8可以包括一个或多个计算装置(未示出)，诸如，个人计算机、膝上型计算机、手持式计算机、蜂窝电话(包括所谓的“智能电话”)、平板电脑、工作站、服务器、交换机、打印机或其他装置。图1所示的网络2的配置仅仅是示例性的。例如，服务提供商网络6可以耦接到任意数量的客户网络8。尽管如此，为了便于描述，图1中仅示出了客户网络8A-8C。

[0034] 多机箱路由器4可以通过包括主要路由引擎以及一个或多个备用路由引擎来提供故障转移。例如，SCC可以包含主要(primary)主控(master)路由引擎和备用主控路由引擎，并且一个或多个LCC可以包含主要本地路由引擎和备用本地路由引擎。主要主控路由引擎可以在将状态信息转发到LCC中的主要本地路由引擎之前，将状态信息传播到备用主控引擎。类似地，主要本地路由引擎在将状态信息转发到其机箱内的消费者组件(其可以被称为“消费者”)之前，将状态信息传播到一个或多个备用本地路由引擎。以这种方式，当在整个多机箱环境中传送状态信息时，多机箱路由器4实施同步梯度。

[0035] 如果主要路由引擎发生故障，则同一机箱中的备用路由引擎取得对该机箱的路由资源和路由功能的控制。此外，由于在将状态信息转发给消费者之前，状态信息传播到备用路由引擎，因此备用路由引擎可以采取在主要路由引擎中断的地方将状态信息转发给消费者。于2003年10月3日提交的题为“Syncronizing State Information Between Control Units”的美国专利号7,739,403描述了用于独立路由器内的同步梯度的技术，并且其通过引用结合于此。于2005年11月16日提交的题为“Push-Based Heirarchical State Propagation within a Multi-Chassis Network Device”的美国专利号7,518,986描述了用于多机箱路由器内的基于推送的状态同步的技术，并且其通过引用结合于此。以这种方式，主要路由引擎和备用路由引擎同步其相应的状态信息，以允许备用路由引擎取得对路由器资源的控制，而不必重新学习状态信息。

[0036] 在主要主控路由引擎将状态信息推送给每个消费者的情况下，每个消费者接收由主要主控路由引擎产生的任何状态信息。随着网络在所提供的服务数量方面变得更大和更复杂，主要主要路由引擎可能同样产生更多关于附加服务的状态信息，这些状态信息必须被潜在的大量消费者消费(特别是在分布式系统的背景下，诸如，具有数百个充当消费者的计算节点的软件定义网络，或者在潜在具有数百个充当消费者的计算节点的大型数据中心中)。生产商组件(其也可以被称为“生产商”)可以指产生状态信息的上述组件中的任一个，诸如，主要主控路由引擎、主要本地路由引擎等。消费者可以表示消费状态信息的上述组件中的任一个，诸如，主要本地路由引擎、接口卡等。

[0037] 在这些情况下，消费者可能变得被状态信息淹没，该状态信息与多机箱路由器4中的消费者的角色可以相关或可以不相关。因此，消费者可能接收到大量的状态信息，这些状

态信息必须被处理以确定这种状态信息是否相关，丢弃消费者为了执行消费者被配置以执行的操作不需要的任何状态信息。由于网络增长并变得越来越复杂(就所提供的服务、执行的协议等而言)，根据消费者被动地接收由生产商产生的所有状态信息(例如，主要主控路由引擎)的推送模型分配状态信息可能不能适应良好。

[0038] 此外，用于传播状态信息的推送模型是以各自执行操作系统的不同实例的生产商和至少一些消费者(例如下级路由引擎，诸如LCC的主要路由引擎和备用路由引擎中的任一个)为前提的。操作系统的每个不同实例都可能需要状态信息的一些子集(在某些情况下，上至及包括全部)。可能发生故障的操作系统的任一个实例(例如，由于硬件故障、功率损失、存储器损坏等)都可能影响多机箱路由器的操作，可能导致分组丢失或分组转发中断。

[0039] 根据本公开中描述的技术，多机箱路由器4可以被配置为在多机箱路由器4的所有计算节点(其可以统称为所有生产商和消费者)上执行操作系统10的单个实例。在本公开中描述的操作系统可被称为分布式操作系统10(“DOS 10”)，因为是在所有计算节点上分布执行。每个计算节点可以自组织、合并，以执行分布式操作系统10的单个实例。计算节点可以包括硬件计算节点(诸如，路由引擎、可以包括专用集成电路的硬件转发单元、以及接口卡)以及由硬件控制单元(例如，一个或多个处理器、一个或多个专用集成电路、现场可编程门阵列等)执行的虚拟计算节点。

[0040] 这样，当多机箱路由器4内的计算节点发生故障时，剩余的计算节点可以继续执行分布式操作系统10的单个实例，而不会可能影响分组转发或多机箱路由器4的其他操作。换言之，支持执行分布式操作系统10的单个实例的计算节点的数量可以扩张和收缩，而在一些情况下不会影响多机箱路由器4的操作。在这个意义上，分布式操作系统可被认为是随着计算节点的数量可能增长或减少(在一定程度上)而具有完全的弹性。

[0041] 在合并以执行分布式操作系统10(其可以被称为“分布式操作系统”)的单个实例之后，计算节点可以在由分布式操作系统提供的执行环境内执行分布式应用套件。在多机箱路由器4的背景下，应用可以包括网络协议，诸如，路由协议、管理协议、管理接口(图形用户界面(GUI)、命令行界面(CLI)等)、通信协议等。

[0042] 分布式操作系统10可以基于多机箱路由器4内的计算节点能力和/或角色，在支持分布式操作系统10的执行的计算节点之中分布应用。分布式操作系统10可以管理状态信息的传播，以支持分布式操作系统10和/或在由分布式操作系统10提供的执行环境内执行的应用的执行。

[0043] 分布式操作系统10可以利用分层命名方案来从生产商向消费者传播状态信息。多机箱路由器4可以建立分层命名方案而不是将每个生产商产生的所有状态信息推送给每一个消费者，以便将对象(其可以指状态信息的分立部分)与分布的范围相关联，这导致对象仅被分发给已请求特定对象的那些消费者。使用分层命名方案，多机箱路由器4内的消费者可以请求任何范围的状态信息，上至并且包括由多机箱路由器4内的任何生产商产生的所有状态信息，并且下至单个对象。有关分层命名方案以及分层命名方案如何可以使用对象洪泛协议(OFP)更有效地传播状态信息的更多信息，可以在于2016年6月30日提交的题为“HIERARCHICAL NAMING SCHEME FOR STATE PROPAGATION WITHIN NETWORK DEVICES”的美国申请序列号15/198,912中找到，如同所阐述的一样，其全部内容通过引用结合于此。

[0044] 在操作时，多机箱路由器4的每个计算节点可以首先执行相同的基础设施，以支持

分布式操作系统10的执行。例如，多机箱路由器4的每个计算节点可以执行内核12，诸如，Unix®内核。因为计算节点尚未合并以支持分布式操作系统10的执行，所以仅在此时，内核12的每个实例的执行视为是“分离”的。在自组织(或换言之，合并)之后，计算节点可以执行单个分布式内核12，至内核12了解由其他计算节点执行的应用和/或其他进程的程度。在所有计算节点上执行统一内核12可以提高可靠性，这是因为内核12可能仅很少被更新，允许发生软件栈11进一步升级而不重新启动(因为内核在升级版本之间可能不会改变)。将内核12与分布式操作系统10的其他方面分离，也可以将内核12的更新周期与分布式操作系统10的其他进程或特征的更新周期分开。

[0045] 在执行内核12的单独实例之后，每个计算节点可以接下来执行OFP 14。如上所述，OFP 14可以在分布式操作系统10的计算节点之间传播状态信息。如上所述，OFP 14可以为状态信息传播提供订阅模型，从而与状态信息传播的推送模型相比，允许潜在更有效地传播状态信息。OFP 14可以允许状态信息传播的多播类型，该多播类型可以可靠地将状态信息同时传送到分布式操作系统10的多个计算节点。

[0046] OFP 14还可以允许自组装，其中，OFP 14提供一种机制，通过该机制发现可用于参与分布式操作系统10的执行的计算节点以及使计算节点互连的链路。OFP 14可以生成表示计算节点和链路的拓扑的图形数据结构，图形数据结构的边线表示使计算节点互连的链路，并且图形数据结构的图形节点表示可用于支持分布式操作系统10的执行的计算节点。图形节点被称为将图形数据结构的节点与支持分布式操作系统10的执行的计算节点区分开的图形节点。除非在别处明确指出或者上下文明确暗示，否则在本公开中对“节点”的引用旨在表示支持分布式操作系统10的执行的节点，而不是图形数据结构的图形节点。OFP 14还可以提供节点可达性服务，以确定节点的活性。

[0047] 在初始化OFP 14之后，多机箱路由器4的每个计算节点可以接下来执行系统纪元管理(SysEpochMan)进程16。SysEpochMan进程16可以组织(到此为止，不同的和独立的)计算节点，以支持单个分布式操作系统10的执行。SysEpochMan进程16还可监控分布式操作系统10，以确保一个或多个计算节点发生故障时的完整性。仅举几例，在计算节点的数量改变、节点间连接的中断、计算节点的组织、和/或计算节点的角色改变的情况下，SysEpochMan进程16可以提供从先前系统状态到新系统状态的转换。

[0048] SysEpochMan进程16可以建立(并维护)Zookeeper®平面(其中，Zookeeper®参考Apache Zookeeper®项目)和OFP域(其可以指OFP域，用于供分布式操作系统10传播特定于分布式操作系统10并且与应用无关的状态信息)。虽然相对于Zookeeper®进行了描述，但是本公开的技术可以相对于任何进程间通信总线或机制来执行。这样，在贯穿本公开，Zookeeper®更通常被称为进程间通信总线18(“IPCB 18”)。

[0049] IPCB 18与OFP 14的不同之处可以在于，OFP 14是异步通信协议(意指OFP可以保证最终的对象传送，而不确保对象的有序传送)，而IPCB 18是同步通信协议(意指IPCB 18可以确保以适当的改变顺序传送，或换言之，所有计算节点按发生变化的顺序接收变化。IPCB 18可以在SysEpochMan进程16内执行，以协调诸如领导者选择(在计算节点内)和命名空间分配的服务。

[0050] 在形成IPCB 18(并且假设OFP 14运行)之后，多机箱路由器4的计算节点可以有效

地彼此通信，以合并并执行分布式操作系统10。计算节点可以接下来执行系统管理器(“SysMan”)进程20，系统管理器进程20协调在由分布式操作系统提供的执行环境内的应用的执行。每个SysMan进程20可以选择SysMan主实例(例如，使用ICPB 18)，该主实例可以负责根据策略引擎(作为一个示例)在特定计算节点上执行应用。

[0051] SysMan主进程可以(例如，经由ICPB 18)将应用决策传送给本地SysMan进程，然后本地SysMan进程作用于应用决策以执行应用。本地SysMan进程监控应用的执行，并向SysMan主进程提供应用的状态，以允许SysMan主进程监控应用的状态。当状态指示应用出现故障时，仅举几例，SysMan主进程可以重新启动应用的执行(通过相同或不同的计算节点)或激活应用的备用实例。

[0052] 多机箱路由器4的计算节点也可以执行分配器进程22，作为分布式操作系统10的一部分。分配器进程22(其也可以称为“分配器22”)可以形成对象守护进程数据存储(DDS)并与各个应用协调，以用于传送状态信息。分配器22可以作为OFP 14的客户端来操作，并且在由不同计算节点执行的分配器之间传送对象。

[0053] 如上所述，分布式操作系统10执行以提供应用可以在其中操作的执行环境。从分布式操作系统的角度来看，计算节点都是统一的，并且只能由每个计算节点执行的应用来区分。应用可以指除了低级驱动器和/或固件以外的、上面没有相对于分布式操作系统10(包括Unix®守护进程)和PFE应用(或换言之，软件)描述的任何进程。

[0054] SysMan 20可以在多个计算节点上分布应用，使用对象来传送与这些类型的分布式应用相关联的状态信息。例如，多机箱路由器4可以执行包括路由协议守护进程(RPD)和一个或多个PFE路由处理器的集合的应用。

[0055] SysMan进程20不将应用绑定到特定硬件，从而允许应用移动性(其也可以被称为“进程移动性”)。SysMan进程20可以在给定计算节点内的处理单元或其他硬件之间或在计算节点之间传递应用，以提供故障恢复、负载平衡和/或服务中系统更新(ISSU)。

[0056] 如上所述，分布式操作系统10首先执行OFP 14，以确定允许由分布式操作系统10的单个实例的计算节点进行的合并和执行的计算节点的拓扑。可以以类似于链路状态协议的方式，发生OFP物理拓扑发现。OFP 14可以通过配置“发现”OFP 14的特定实例耦接到的链路。即，网络管理员或其他运营商可以利用OFP 14的每个特定实例所耦接到的链路来配置OFP 14。OFP 14使用通告协议发现计算节点，通过该通告协议，每个计算节点在该计算节点连接到的每个链路上周期性地多播计算节点标识。

[0057] OFP 14将每个计算节点分类为主要计算节点和次要计算节点。管理员可以将计算节点配置为主要或次要，其中，主要计算节点的实例可以包括路由引擎(或换言之，支持控制面板的计算节点)，并且次要计算节点的实例可以包括线卡(或换言之，支持转发平面的计算节点)。在一些情况下，主要计算节点可以指与次要计算节点相比具有增强的处理和/或存储器能力的任何计算节点。考虑到增强的处理和存储器能力，OFP 14可以尝试将尽可能多的处理卸载到主要计算节点。

[0058] 主要OFP计算节点可以将通告发送给参与分布式操作系统10的所有计算节点(意指所有主要和次要的OFP计算节点)。次要OFP计算节点可以将通告发送到所有的主要OFP计算节点(并且在一些示例中，非所有的次要OFP计算节点)。尽管被描述为没有向主要OFP计算节点传输通告，但是在一些示例中，次要OFP计算节点可以向一个或多个次要OFP计算节

点传输通告。

[0059] 接收到通告的每个OFP计算节点配置物理拓扑图形数据结构，以将通告计算节点标识为邻居。假设次要计算节点仅向主要OFP计算节点传输通告，则次要OFP计算节点不能彼此成为邻居，因为次要OFP计算节点从未从通过其建立邻居关系的另一次要OFP计算节点接收到通告。OFP14基于通告构建表示通过链路彼此互连的主要和次要的计算节点的拓扑的图形数据结构。

[0060] 在OFP 14构建表示主要和次要计算节点的拓扑的图形数据结构之后，SysEpochMan 16可以从被配置为作为纪元管理器执行的那些计算节点之中选择纪元管理器主控(epoch manager master)。作为一个示例，网络管理员可以将能够执行进程间通信总线18(IPCB，例如，Zookeeper®)的计算节点配置为纪元管理器。所选择的纪元管理器主控可以选择一个或多个纪元管理器(包括所选择的纪元管理器主控)充当纪元管理器。然后每个纪元管理器可以执行IPCB 18。

[0061] IPCB 18形成服务器和客户端的网络。这些服务器可以被称为ICPB总体(ensemble)。IPCB 18可以利用群体系统，在该系统中，大多数服务器(例如，多于(N/2)+1，其中，N表示服务器/纪元管理器的数量)连接并且用于IPCB 18继续成功操作。IPCB客户端表示利用IPCB 18的计算节点。IPCB客户端可以与任何IPCB服务器交互，以利用IPCB 18。利用IPCB 18，IPCB客户端可以与共享文件系统交互，以将数据写入共享文件系统和/或从共享文件系统读取数据，同时也能够配置有关共享文件系统的变化的通知。以这种方式，为了执行分布式操作系统10的目的，这些技术可以允许分离的(或换言之，单独的)计算节点合并。

[0062] 在成功启动分布式操作系统10时，分布式操作系统10可呈现另一OFP域，以由应用用于从生产商向消费者传播状态信息。例如，多机箱路由器4的计算节点可以同步用于分布式操作系统10、应用或多机箱路由器4的其他元件的状态信息。特别地，每个计算节点可以使存储多个对象的相应数据结构实例化，其中，每个对象定义例如用于分布式操作系统10和/或用于一个或多个应用的状态信息的至少一部分。计算节点可以通过执行OFP 14，以根据OFP域来使相应的数据结构同步。此外，计算节点可以使用同步的数据结构来配置例如其自身和/或计算节点的其他组件。

[0063] 基数树(radix trie)是由其关键字(key)构成的树，对于基数树，每个内部树节点有至少两个子。为了定位具有特定关键字的树节点，通过从左手边开始检查关键字的内容来走树。基数树是最小的树，因为没有内部树节点而只有一个子。帕氏树(Patricia Trie)是基数树的一种特定形式。

[0064] 在一些示例中，计算节点可以将用于存储多个对象的数据结构实例化为树数据结构，例如，基数树。例如，执行分布式操作系统10的各种实例的计算节点可以实例化以分层方式(即根据树数据结构)排列的一个或多个主题。分层排列的主题可以具有各种层次的范围，主题位于层次中的另一主题之上，包括发布到位于底层主题上面的主题之下的主题的任何状态。计算节点因此可以实例化树数据结构，以存储分层排列的主题，其中，执行分布式操作系统10的相应实例的计算节点可以形成树数据结构的到主题的树节点。

[0065] 例如，主题“/a”可以是主题“/a”、“/a/b”和“a/b/c”的聚合。作为另一示例，主题“/a/b”可以是主题“a/b”和“a/b/c”的聚合。因此，树数据结构的第一树节点可以对应于主题

“/a”，第一树节点的第一叶树节点可以对应于主题“/a/b”，并且第一叶树节点的第二叶树节点可以对应于主题“/a/b/c”。以这种方式，可以从树数据结构的树节点和来自该节点的叶树节点获得主题串。主题串可以对应于主题的串表示，在这种情况下恰好是主题层次本身。在一些示例中，类似于树数据结构，分层排列的主题将仅具有在根主题树节点下具有多个分层结构的一个根主题(其可以被称为“根主题树节点”)。

[0066] 节点的其他节点、应用或组件可充当这些主题的消费者。一旦这些主题被实例化，消费者可以接收对本地主题数据库的更新，通知消费者新的主题。消费者然后可以订阅新的主题，使得发布到该主题的对象仅被分配给订阅的消费者。消费者然后可以消费对象来更新本地状态信息，而不必过滤或以其他方式丢弃与消费者的操作无关的对象。

[0067] 生产商可以通过与应用基础设施的交互来实例化层次结构内的主题。应用基础设施可以用一个或多个32位范围ID(其统称为范围向量)来标记每个主题，该范围ID标识相应的标记对象被传送到的范围。每个消费者订阅一个或多个范围ID(经由对对应的主题的请求)，并且应用基础设施自动将具有对应的范围ID的对象传送给请求这些主题的消费者。如图3所示，相对于多机箱路由器4的单个计算节点，更详细地描述负责将范围ID映射到对象以及分配对象的各个单元。

[0068] 在OFP中，树数据结构的叶子表示单独的消息片段，并且关键字是片段ID。因此，任何内部树节点表示片段ID前缀，并且因此表示片段ID的范围。根树节点可以表示零长度前缀以及所有可能片段的“范围”。每个树节点携带表示前缀覆盖的所有片段的摘要的散列(hash)值。

[0069] 叶树节点是简并示例，摘要是片段的贡献，并且片段ID前缀是片段ID本身。根据OFP，计算节点200(或其处理单元)计算来自之前描述的(逻辑时钟、校验和)元组的贡献，并根据其片段ID将对应的树节点定位在树数据结构中。通过将对片段本身的引用添加到叶树节点，树数据结构也可以用于查找片段。

[0070] 基数树节点的最大扇出是VS，其中，V是符号的可能值的数量，并且S是符号计数。在OFP中，符号是单个位(片段ID被视为位串)，因此可以选择S的值以提供适当的扇出量。严格二叉树将具有V＝2和S＝1，导致非常深的树。对于OFP，S通常是大于1的小值(例如，4)，这使得树有点分枝且不那么深。

[0071] 树数据结构可以是不可变的。树数据结构的不变性可以有助于缩放，因为这意味着所有操作(包括任意子树的提取)都可以在Log时间内完成，除了遍历(需要N*log(N)时间)外。OFP 14可以将树数据结构设置为不可变的，这可以提高可伸缩性。

[0072] 树数据结构的树节点可以表示“摘要”(其类似于校验和)的层次。摘要可以包括例如标量值(例如，修改的Fletcher-64校验和)，该标量值表示由树数据结构的相应的一个树节点可访问的树数据结构的叶树节点存储的内容。支持分布式操作系统10的执行的节点可以将消息片段存储在由相应片段标识符(片段ID)排列的树数据结构中。OFP 14可以将消息分成一系列片段，每个片段适配到单个分组中。OFP 14可以使用具有包括元组(范围、消息ID、片段号)的片段ID以及来自原始分离消息的逻辑时钟值来标记片段。OFP的可靠性模型对单独片段进行操作(从而减小了丢失分组的影响)。这样，支持分布式操作系统10的执行的计算节点可以将消息分成组成片段，并且将每个片段作为树节点存储在由相应片段的片段ID排列的树数据结构中。

[0073] 此外，支持分布式操作系统10的执行的计算节点可以形成树数据结构的内部树节点，以表示片段ID的块体(以前缀的形式)并且在其表示的块体中包括表示所有片段的摘要。因此，表示零长度前缀的树数据结构的根包括覆盖拓扑中的所有消息的摘要。这样，确定两个树节点具有相同拓扑内容的成本被降低到O(1)(只要内容相同)。

[0074] 每当分布式操作系统10的一个计算节点修改一个消息片段时，分布式操作系统10的一个计算节点也递增地更新所有消息片段的祖先的摘要，返回到树数据结构的根。

[0075] 以这种方式，分布式操作系统10的两个或更多个计算节点可以通过比较相应树数据结构的树节点的相应摘要，来使其相应的树数据结构同步。当树数据结构的对应树节点的摘要匹配时，分布式操作系统10的计算节点可以确定树数据结构是同步的。然而，当对应的树节点的摘要不匹配时，分布式操作系统10的计算节点可以确定树数据结构不同步。因此，分布式操作系统10的计算节点可以交换消息(例如，以消息片段的形式)，以使相应树数据结构同步。因此，当树数据结构在每个树数据结构内具有树节点的共同排列和互连时并且当树数据结构的对应树节点的摘要匹配时，两个树数据结构可被描述为同步的。

[0076] 例如，分布式操作系统10的计算节点可以首先确定其相应的树数据结构的两个对应的树节点不同步。然后，分布式操作系统10的计算节点可以确定相应树数据结构的两个树节点中的哪一个包括更高(即，更近)的逻辑时钟值。具有更近的逻辑时钟值的树数据结构的树节点可以被认为是最新的，并且因此是正确的。相应地，拥有树数据结构的具有更近的逻辑时钟值的树节点的分布式操作系统10的计算节点，可以将用于树数据结构的对应的消息或消息片段发送到分布式操作系统10的其他计算节点。分布式操作系统10的其他计算节点可以使用消息或消息片段来更新其对应的树数据结构，从而使树数据结构的至少这些分支同步。

[0077] 分布式操作系统10的计算节点可以进一步添加、修改或删除消息片段。为了添加或删除消息片段，分布式操作系统10的计算节点修改相应的树数据结构，以向树数据结构添加或从树数据结构删除对应的树节点。为了修改消息片段，分布式操作系统10的计算节点更新树数据结构的适当树节点的内容。此外，响应于添加、修改或删除消息片段，分布式操作系统10的计算节点将对应的树数据结构从叶树节点行进到根，沿途递增地更新树数据结构的树节点的摘要。由于任何树节点的摘要值都是贡献，所以从其父摘要中减去旧摘要(如上所定义)，并添加新值，并且该进程朝着根向上递归。

[0078] 在树数据结构是基数树并且摘要是Fletcher-64校验和的示例中，添加或者删除叶树节点可以导致创建或者删除内部树节点。不存在的树节点的贡献可以是零(由于使用Fletcher)，使得当创建或销毁树节点时，使用该值。

[0079] 在这些示例中，更新树数据结构的最坏情况成本是O(logF N)，其中，F是最大树节点扇出，N是消息片段的数量。实际上，这可以是相当小的，伴随一百万个对象和16个扇出，成本是O(5)，对于1600万个对象，是O(6)等。以这种方式，这些技术可以在各种计算节点之间有效地维护状态同步，以用于执行分布式操作系统10或应用或分布式操作系统10和应用这两者。

[0080] 在形成群体并建立客户端可以通过其与共享文件系统交互并从而执行分布式操作系统10以促进状态信息的交换和同步的IPCB 18之后，IPCB 18可以监控IPCB服务器，以确定在多个计算节点中的一个或多个之间的连接是否发生故障。例如，当IPCB纪元管理器

发生故障或者链路故障(通常可以称为“连接故障”)时,剩余的IPCB纪元管理器可以确定纪元管理器的群体是否存在。

[0081] 剩余的IPCB纪元管理器可以通过比较仍然操作的纪元管理器的数量(由变量“N”表示)大于、或者大于或等于连接故障阈值(例如(N/2)+1)来确定纪元管理器的群体是否存在。连接故障阈值也可以被称为“群体阈值”。当剩余的纪元管理器的数量超过连接故障阈值时,剩余的纪元管理器可以维护群体并继续操作,可能向群体添加在群体形成期间没有当选为纪元管理器的新纪元管理器。当剩余的纪元管理器的数量没有超过连接故障阈值时,剩余的纪元管理器可以重新启动分布式操作系统10(其可以不需要重新启动多机箱路由器4或者内核12,而是仅重新启动软件栈11中的内核12之上的那些层中的一个或者多个,诸如,OFP 14、SysEpochMan 16、IPCB 18、SysMan 20和/或分配器22)。

[0082] 以这种方式,多机箱路由器4的分布式操作系统10可以从各种不同类型和能力的多个不同计算节点合并。OFP 14可以执行以发现计算节点拓扑,从而允许IPCB 18形成,以建立群体,通过该群体来确保足够的资源,以继续分布式操作系统10的成功执行。群体可以确保足够的资源可用于允许状态信息的成功传播,同时,如下面更详细描述的那样,还允许通过其克服计算节点拓扑被分成两个不同的执行环境的裂脑情况的机制。

[0083] 图2是示出被配置为根据本公开中描述的技术进行操作的示例性多机箱路由器120的框图。多机箱路由器120通过网络在网络装置之间路由数据分组。在该示例中,多机箱路由器120包括作为中央控制节点操作的四个基本相同的LCC 128A-128D(“LCC 128”)和SCC 122。在其他实施方式中,多机箱路由器可以包括更多或更少的LCC。SCC 122为多机箱路由器120提供集中式交换和控制。LCC 128使用IFC组134A-134D(“IFC 134”)提供到网络的接口。

[0084] SCC 122包括交换结构124和主控路由引擎126。虽然在图2的示例中未示出,但是当多机箱路由器120被配置为高可用性路由器时,SCC 122可以包括备用主控路由引擎。交换结构124在LCC 128的交换结构125之间提供背侧连接,即与网络分离的连接。主控路由引擎126的功能包括维护路由信息以描述网络的拓扑,并使用该信息来导出转发信息库(FIB)。经由通过缆线137与本地路由引擎130的通信通过在LCC 128中安装FIB,路由引擎126在整个多机箱路由器120中控制分组转发。一个LCC 128的FIB可以与其他LCC 128和SCC 122的FIB相同或不同。因为缆线137在SCC 122与LCC 128之间提供专用连接(即与由缆线136提供的数据分组转发连接分离),所以LCC路由引擎130中的FIB可以被更新,而不中断多机箱路由器120的分组转发性能。

[0085] LCC 128各自包含一个本地路由引擎130A-130D(“路由引擎130”)、一个交换结构125A-125D(“交换结构125”)、示出为PFE 132A-132D(“PFE 132”)的至少一个分组转发引擎(PFE)以及一个或多个IFC 134。在一些示例中,当多机箱路由器120被配置为提供高可用性时,除了可以在高可用性配置中称为主要本地路由引擎130的一个本地路由引擎130之外,LCC 128还可以包括一个备用本地路由引擎。

[0086] 多机箱路由器120以以下方式执行路由功能。首先由一个IFC 134(例如,134B)从网络接收传入数据分组,该IFC 134将传入数据分组引导到一个PFE 132(例如,PFE 132B)。然后,PFE使用由主要本地路由引擎(例如,路由引擎130B)提供的FIB来确定对于数据分组的适当路由。如果数据分组指向与最初接收分组的一个IFC 134相关联的出站链路,则PFE

将分组转发到出站链路。以这种方式，由从网络接收分组的相同PFE发出的分组旁路交换结构124和交换结构125。

[0087] 否则，PFE将数据分组发送到交换结构125，其中，该分组被引导到交换结构124并且遵循到一个其他PFE 132(例如，PFE 132D)的路由。该PFE(例如，PFE 132D)经由一个IFC 134(例如，IFC 134D)通过网络发送数据分组。因此，由一个LCC 128接收的传入数据分组可以由另一LCC 128发送到其目的地。以与本公开中描述的技术一致的方式操作的其他多机箱路由器可以使用不同的交换和路由机制。

[0088] 本地路由引擎130控制和管理LCC 128，但是服从于SCC 122的主控路由引擎126。例如，在从主控路由引擎126接收到状态信息更新之后，本地路由引擎130使用分级排序和时间链接的数据结构，将状态信息更新转发给在LCC 128上的消费者。例如，从本地路由引擎130接收状态信息更新的消费者包括PFE 132和IFC 134。本地路由引擎130还将由主要主控路由引擎126导出的FIB分配给PFE 132。

[0089] 路由引擎126和130可以根据从一个或多个计算机可读介质提取的可执行指令来操作。这种介质的示例包括随机存取存储器(RAM)、只读存储器(ROM)、非易失性随机存取存储器(NVRAM)、电可擦除可编程只读存储器(EEPROM)、闪存等。多机箱路由器120的功能可以通过利用一个或多个处理器、分立的硬件电路、固件、在可编程处理器上执行的软件或其组合执行计算机可读介质的指令来实现。

[0090] 如上所述，节点可以包括路由引擎126、路由引擎130、PFE 132和IFC 134。链路可以包括交换结构124和缆线136和137以及为了便于说明而示出但未列举的其他缆线。各种节点可以执行下面相对于图3中所示的多机箱路由器120的单个节点更详细描述的技术的方面。

[0091] 图3是示出被配置为根据本公开中描述的技术的各个方面进行操作的多机箱路由器120的示例性计算节点200的框图。作为示例，计算节点200可以表示路由引擎126、路由引擎130中的一个、交换卡机箱122或线卡机箱128中的一个。

[0092] 如图3所示，计算节点200执行能够与计算节点200的硬件交互的内核12。一旦内核12运行，计算节点200就可以执行OFP 14，通过该OFP14确定在多机箱路由器120内执行的计算节点的拓扑202。拓扑202可以表示上述图形数据结构，该图形数据结构包括表示多机箱路由器120的计算节点的图形节点和使图形节点互连的表示使多机箱路由器120的计算节点互连的链路的边线。

[0093] OFP 14可以通过接收通告204来发现或以其他方式确定拓扑202。OFP 14可以从支持执行分布式操作系统10的每个其他计算节点(作为状态信息的生产商或消费者)接收通告204。每个公告204可以指定计算节点以及直接耦接到该计算节点的一个或多个链路。OFP 14可以配置(例如，通过网络管理员)有直接耦接到计算节点200的链路。OFF 14可以从通告204和链路206构建拓扑202。根据本公开的技术，OFP 14还包括表示状态数据结构(诸如，树数据结构)的状态208。

[0094] OFP 14还可以从链路206生成通告204，经由链路206所标识的链路发送所生成的一个通告204，使得由相邻计算节点执行的OFP 14可以类似地生成拓扑202。与链路状态路由协议一样，OFP 14操作以在每个计算节点处(或者在一些情况下，仅在主要计算节点处)形成拓扑202的本地副本。OFP 14可以经由通告204对计算节点200检测到的拓扑202的变化

(例如,计算节点或链路断开)进行洪泛,由此允许拓扑202在支持执行分布式操作系统10的每个计算节点200处保持同步。OFP可以将拓扑202(经由应用编程接口API)暴露给SysEpochMan 16。

[0095] 在初始化期间,SysEpochMan 16可以首先订阅OFP域0内的EpochManagement范围,并且假设SysEpochMan 16被配置为具有纪元管理器(EM或Em)能力,订阅OFP域0内的EmCapableNodes范围。SysEpochMan 16可以首先将纪元管理器对象210发布到OFP域0内(其如上所述由OFP 14形成,以供分布式操作系统10的底层基础设施(诸如,OFP 14、SysEpochMan 16、IPCB 18等)使用)。纪元管理器对象210指示计算节点200是否已被配置为能够充当纪元管理器,并且纪元管理器优先级被配置为用于计算节点200充当纪元管理器。与较低的纪元管理器优先级相比,较高的纪元管理器优先级指示计算节点200更可能被选为纪元管理器。因而,纪元管理器优先级允许网络管理员将纪元管理器功能偏向或远离特定的计算节点。

[0096] 纪元管理器对象210还可以包括硬件主控指示,硬件主控指示指示计算节点200是否拥有硬件主控权,其中,可以在两个具有纪元管理器能力的节点系统中使用这种信息来确定是否存在群体。纪元管理器对象210还可以包括指示用于充当纪元管理器主控的计算节点200的提名的主控标识符(ID)。纪元管理器对象210还可以指示主控优先级,该主控优先级可以指示用于纪元管理器主控选择的计算节点200的优先级。像纪元管理器优先级一样,与更低的纪元管理器主控优先级相比,更高的纪元管理器主控优先级指示计算节点200更可能被选为纪元管理器主控。因而,纪元管理器主控优先级允许网络管理员将纪元管理器主控功能偏向或远离特定的计算节点。

[0097] 纪元管理器对象210还可以指定纪元号,纪元号可以指示计算节点200先前参与的分布式操作系统10的纪元。纪元可以指在一段时间内运行的分布式操作系统10的版本。纪元号允许在分布式操作系统10的最近运行版本上合并计算节点200。下面更详细地讨论纪元。

[0098] 纪元管理器对象210还可以包括是否已经将计算节点200选择为纪元管理器的指示以及是否已经将计算节点200选择为纪元管理器主控的指示。另外,纪元管理器对象210可以包括计算节点200是否已成功加入纪元(其由于将数据成功写入IPCB服务器中的一个而合格)的指示以及计算节点200是否成功地用作纪元管理器(其由于将数据成功写入IPCB服务器中的一个而合格)的指示。

[0099] 此外,纪元管理器对象210可以包括重新启动请求,重新启动请求请求重新启动分布式操作系统10,而保留当前纪元管理器组或重置纪元管理器组。纪元管理器对象210还可以包括设置系统中预期的具有纪元管理器能力的计算节点的最大数量的指示,而零值表示没有设置最大值。

[0100] 为了重申以上描述,纪元管理器对象210可以包括以下内容:

```
object EmNode {
        NodeID id; //发布者的节点 ID
        Int nonce; //随机数
        Boolean emCapable; //如果具有 EM 能力，则为真
        Int emPriority; //EM 选择的优先级
        Boolean hwMaster; //如果是硬件主控，则为真
        NodeID masterId; //提名主控的 ID，或 0
        Int masterPriority; //EM 主控选择的优先级
        SysEpoch epoch; //本地系统纪元，或 0
        Boolean epochManager; //如果节点是纪元管理器，则为真
        Boolean emMaster; //如果节点是纪元管理器主控，则为真
        Boolean epochUp; //如果纪元出现，则为真
        Boolean managerUp; //如果纪元管理器出现，则为真
        Enum restartRequest; //重新启动请求
        Int maxEmCapableNodes; //具有 EM 能力的节点的最大数量#，
    或 0
}
```

[0101]

[0102]

[0103] 每当节点的内容改变时，每个节点都在OFP中更新其对象。所有具有EM能力的计算节点都订阅这些对象。

[0104] 这些字段具有以下语义：

[0105]

id	发布节点的节点 ID。该值由 OFP 提供。
nonce	当节点重新启动时由 OFP 生成的随机数。该值与在 OFP 可达性协议中发送的值进行比较。如果不同，则意味着节点已经重新启动，并且应忽略对象。如果发布节点重新启动，这有效地使对象不可见。
emCapable	如果节点具有纪元管理器能力，则设置为真。
emPriority	发布节点对于纪元管理器选择的优先级，如果节点不具有 EM 能力，则为 0。在选择纪元管理器的组时更高优先级的节点会受到青睐，从而提供将纪元管理器功能偏向特定节点的方法。
hwMaster	如果节点拥有硬件主控权（如果其存在于此节点类型中），则设置为真，否则设置为假。这用于具有 EM 能力的两节点系统，以确定是否存在群体。
masterId	发布节点对于纪元管理器主控的提名的节点 ID，如果节点尚未决定或不具有 EM 能力，则为 0。
masterPriority	发布节点对纪元管理器主控选择的优先级。在 EM 管理器选择时更高优先级的节点将受到青睐，从而提供将 EM 管理器功能偏向特定节点的方法。
epoch	发布节点对系统纪元的理解，如果节点还没有加入纪元，则为 0。
epochManager	如果发布节点已经被选为纪元管理器，则为真。

[0106]

emMaster 如果发布节点已经被选为纪元管理器主控，则为真。

epochUp 如果发布节点已成功加入纪元（由于将数据成功地写入 Zookeeper），则为真。

managerUp 如果发布节点用作纪元管理器（由于已通过发布节点的服务器将数据成功地写入 Zookeeper 中），则为真。

restartRequest 节点的重新启动请求。可能的值是 None、Restart 和 ResetManagers。这用于用户请求的重新启动（而不是由于群体丢失而强制重新启动）。重新启动的值跨重新启动期间保留先前的 EM 管理器组，并且 ResetManagers 将其重置。后者用于在故障导致具有 EM 能力的节点的不可恢复的丢失使得不能满足先前管理器组的群体之后，允许重新启动（否则系统将永远不会恢复）。

maxEmCapableNodes 设置为系统中预期的具有 EM 能力的节点的最大数量。在单 EM 节点系统中设置为 1，在双 EM 节点系统中设置为 2，并且在其他情况下设置为 0。

[0107] 包括计算节点200的每个节点然后将其纪元管理器对象210设置为如下：

[0108]

local.id 节点 ID（由 OFP 提供）。

local.nonce 随机数（由 OFP 提供）。

local.emCapable 如果节点具有纪元管理器能力，则设置为真。

local.emPriority 发布节点对于纪元管理器选择的优先级，如果节点不具有 EM 能力，则为 0。

local.masterId 0。

local.masterPriority 发布节点对于纪元管理器主控选择的优先级，如果该节点不具有 EM 能力或不希望成为 EM 主控，则为 0。

local.epoch 0。

local.epochManager 假。

[0109]

local.emMaster 假。

local.epochUp 假。

local.managerUp 假。

local.restartRequest 无。

local.maxEmCapableNodes 0、1 或 2，取决于硬件配置（单节点和双节点系统中的所有节点期望知道这个事实）。

[0110] 假设计算节点200已经被配置为能够作为纪元管理器进行操作，则SysEpochMan 16接收每个发布的纪元管理器对象210。SysEpochMan 16可以从纪元管理器对象210确定能够充当纪元管理器的哪个计算节点来充当纪元管理器主控。在等待某个时间段(表示为“EmWaitTime”)，以允许对象到达(并且避免最近重新启动的计算节点立即将自身选为纪元管理器主控)之后，SysEpochMan 16可以基于每个纪元管理器对象210的纪元管理器主控优先级来确定哪个计算节点充当纪元管理器主控。在继续执行对象事件进程之前，SysEpochMan 16也可以删除任何IPCB状态信息。

[0111] 包括计算节点200的SysEpochMan 16的所有计算节点可以在启动时或者每当其订阅的任何EmNode对象210或EmMaster对象212(这是指代纪元管理器主控对象212的另一种方式)(包括其自身)改变时，执行对象事件进程。在更新EmNode对象的本地副本时，不具有EM能力的计算节点不执行对象事件进程，因为它们未订阅EmNode对象。

[0112] 当计算节点更新对象事件进程中的对象时，计算节点再次执行对象事件进程(因为其自身的对象已经改变)，只要其订阅了该对象被发布到的范围。重复这一点，直到没有对象被更新。当计算节点在下面的进程中重新启动时，计算节点退出对象事件进程。

[0113] 在对象事件进程的早期，SysEpochMan 16选择单个EmMaster对象210(如果存在这种对象210)。对象事件进程可以引用epochState字段，该字段可以设置为以下中的任一项：

[0114]

EpochStart 初始状态。从某种意义上说，这不是纪元的状态，而是当选的 EM 主控的状态，该主控试图决定如何前进。纪元管理器主控已经当选并且正在等待具有 EM 能力

[0115]

	的节点的群体到达。EM 主控不会在此状态下发布 EmMaster 对象，使得任何旧的 EmMaster 对象继续保留。当具有 EM 能力的节点的群体形成时，转换到状态 EpochInit。如果任何节点请求系统重新启动，则转换为状态 EpochFail。
EpochInit	纪元正在初始化。在 OFP 中可达到具有 EM 能力的节点的群体，但并非所有选择的纪元管理器都已准备就绪。当选为纪元管理器的节点的群体准备就绪时，转换到状态 EpochRunning。如果具有 EM 能力的节点的群体发生故障或任何节点请求系统重新启动，则转换为状态 EpochFail。
EpochRunning	纪元出现。较高层已经启动并且系统正在运行。当 EM 主控决定改变该组 EM 节点时，转换到状态 EpochReconfig。如果 EM 群体发生故障或任何节点请求重新启动系统，则转换为状态 EpochFail。
EpochReconfig	该组 EM 节点正在由 EM 主控重新配置，但尚未完成。尽管 Zookeeper 状态变化失速，但系统继续运行。当选择的纪元管理器的群体准备就绪时，转换到状态 EpochRunning。如果选择的纪元管理器的群体发生故障或任何节点请求系统重新启动，则转换为状态 EpochFail。
EpochFail	纪元已由于丢失群体或全系统重新启动而发生故障。将创造一个新的纪元，并且将毁灭这个纪元。

[0116] 对象事件进程可以相对于所选对象如下进行操作：

[0117] (执行本地家务:)

[0118] |如果存在一个或多个现有EmNode对象，对于该对象，local.id==remote.id并且local.nonce!=remote.nonce，则删除对象(本地节点已重新启动)。

[0119] |如果存在一个或多个现有EmNode对象(本地对象除外)，对于该对象，local.id==remote.id并且local.nonce==remote.nonce，则重新启动节点(因为发生错误，需要重新启动)。

[0120] (选择EmMaster对象:)

[0121] |如果存在至少一个EmMaster对象：

[0122] ||选择最佳EmMaster对象。因为可能有不止一个(由于OFP的异步性质)，所以优选master.epochState!＝EpochFail的对象，然后优选具有master.epochPreference的最高值的对象，并且然后优选具有master.masterId的最高值的对象。如果存在不止一个，这可能导致所有节点会聚到单个EmMaster对象并且选择“最佳”纪元。并且如果有任何未发生故障的纪元，则忽略发生故障的纪元。

[0123] (设置/验证系统纪元:)

[0124] ||如果local.epoch＝＝0:并且master.epochState!＝EpochFail

[0125] |||设置local.epoch＝master.masterEpoch。

[0126] ||否则，如果local.epoch!＝0:(已经是纪元的一部分)

[0127] |||如果local.epoch!＝master.masterEpoch，则重新启动节点。这意味着纪元已经改变了。

[0128] |||如果master.epochState＝＝EpochFail，则重新启动节点。这意味着纪元已经发生故障，并且系统正在重新启动。

[0129] |||如果master.epochState＝＝EpochRunning并且local.epochUp＝＝True且上层尚未运行，则利用EmMaster对象中的OFP域和Zookeeper参数启动上层。这意味着系统已经出现。

[0130] (更新用于检测Epoch Up的这组EM管理器:)

[0131] ||如果local.epochUp＝＝False并且local.epoch!＝0:

[0132] |||将以前的任何Zookeeper客户端会话重置为master.managers中的一组节点(该组管理器可能已改变)。

[0133] |||经由master.zkClientPort端口，向作为服务器的master.managers中的这组节点打开Zookeeper客户端会话。

[0134] |||向“/SysEpochMan/EpochRunning/<id>”发布Zookeeper写入，其中，<id>是发布者的节点ID的文本表示。如果并且当此写入完成时，将导致Epoch Up事件。

[0135] |||向“/SysEpochMan/SystemEpoch/”发布Zookeeper getChildren监视(watch)。如果并且当读取完成时，将导致Zookeeper Epoch事件。

[0136] (所有节点查看是否已经丢失群体:)

[0137] |如果local.epochUp＝＝True且EM节点的群体发生故障(见下面的部分8.4.11)，则重新启动节点。如果local.emMaster＝＝True(此节点是EM Master)，则设置master.epochState＝EpochFail并在重新启动之前发布更新的EmMaster对象。这意味着网络已经分区，或者有太多EM节点发生故障且必须放弃纪元，并且我们需要EM Master来以信号发送这个事实。

[0138] (不具有EM能力的节点在这里退出，想要正常关闭但是当前是纪元管理器的节点例外，这种节点继续其EM角色，直到被EM主控解散:)

[0139] |如果local.emCapable＝＝False并且local.epochManager＝＝False，则退出对象事件进程(该节点不具有EM能力，或者由于正在关闭而被解除了作为纪元管理器的职责)。

[0140] (所有具有EM能力的节点都执行EM主控权选择:)

[0141] |将local.masterId设置为在remote.masterId中报告自己ID的节点的ID,其最高值为remote.masterPriority,然后设置为最低节点ID。如果没有这种节点,则选择remote.emCapable==True的节点的ID,其最高值为remote.masterPriority,然后选择最低节点ID。如果没有这种节点,则使用0。(注意,如果节点变得无法访问,则其EmNode对象将隐藏,因此将只考虑可达的节点。)

[0142] (所有具有EM能力的节点都会查看其EM状态是否已经改变:)

[0143] |如果local.epochManager==False并且master.managers包含(local.id,local.nonce):(成为纪元管理器)

[0144] ||设置local.managerUp=False。

[0145] ||设置local.epochManager=True。

[0146] ||将master.managers中的这组可达服务器写入IPCB服务器配置文件。

[0147] ||清除任何本地持久的IPCB服务器状态。

[0148] ||在master.zkServerPort和master.zkElectionPort中指定的端口上启动本地IPCB服务器。如果master.managers的大小为1,则以独立模式启动IPCB;否则,以复制模式启动。

[0149] ||经由master.zkClientPort端口向作为服务器的节点local.id打开IPCB客户端会话,并且向"/SysEpochMan/ManagerUp/<id>"发布IPCB写入,其中,<id>是发布者的节点ID的文本表示。如果并且当此写入完成时,将导致管理器出现(Manager Up)事件。

[0150] |否则,如果local.epochManager==True并且master.epochState!=EpochReconfig且master.managers不包含(local.id、local.nonce):(不再是纪元管理器)

[0151] ||设置local.managerUp=False。

[0152] ||设置local.epochManager=False。

[0153] ||关闭任何本地IPCB服务器。

[0154] ||关闭对于管理器出现事件的任何客户端会话。

[0155] (如果合适的话,则在独立模式和复制模式之间切换IPCB:)

[0156] |否则,如果local.epochManager==True并且master.managers包含(local.id,local.nonce):(已经是纪元管理器)

[0157] ||如果master.managers的大小是1并且IPCB 18正在以复制模式运行:

[0158] |||将master.managers中的服务器写入IPCB服务器配置文件。

[0159] |||在master.zkServerPort和master.zkElectionPort中指定的端口上,以独立模式重新启动本地IPCB服务器。

[0160] ||否则,如果master.managers的大小大于1并且IPCB正在以独立模式运行:

[0161] |||将master.managers中的该组可达的服务器写入IPCB服务器配置文件。

[0162] |||在master.zkServerPort和master.zkElectionPort中指定的端口上,以复制模式重新启动本地IPCB服务器。

[0163] (如果合适的话,则执行EM主控职责:)

[0164] |如果local.masterId==local.id:(这个节点是或刚成为主控)

[0165] ||如果任何local.emMaster==False:(成为主控)

[0166] |||如果任何remote.masterId!=local.id,退出对象事件进程。这意味着本地节

点的选择还不是一致的。

[0167] |||如果master.epochState＝＝EpochFail并且master.managers不为空，且master.managers中的节点的群体(参见部分8.4.10)不可达(忽略随机数值)，则退出对象事件进程。这意味着我们可能已被分区，并且因此不想推进纪元，以免造成裂脑。

[0168] |||如果任何EmMaster对象与master.masterId＝＝local.id一起存在，则将其删除(从此节点清除旧的EmMaster对象)。

[0169] |||设置local.emMaster＝True。

[0170] |||根据部分8.4.12，初始化本地EmMaster对象。

[0171] ||根据部分8.4.13，更新EmMaster状态。

[0172] ||如果master.epochState!＝EpochStart：

[0173] |||如果已改变，则更新OFP中的EmMaster对象。

[0174] |||如果任何EmMaster对象与master.masterId!＝local.id一起存在，则将其删除(从其他节点清除旧的EmMaster对象)。

[0175] 所选的纪元管理器主控(为了解释，假设这是计算节点200)可以在被选择时将纪元管理器主控对象212发布到OFP域0内。纪元管理器主控对象212可以包括以下信息。

[0176] 对象EmMaster{

[0177] NodeIDmasterId;//发布者(EM主控)的节点ID

[0178] SysEpoch masterEpoch;//全球系统纪元

[0179] Int epochPreference;//纪元偏好

[0180] Int zkClientPort;//Zookeeper客户端端口

[0181] Int zkServerPort;//Zookeeper服务器端口

[0182] Int zkElectionPort;//Zookeeper领导选择端口

[0183] Int ofpDomain;//OFP域ID

[0184] Enum epochState;//纪元状态

[0185] (NodeID,Int)managers[];//选择的EM及其随机数

[0186] (NodeID,Int)oldManagers[];//以前的EM及其随机数

[0187] Int maxManagerCount;//预计最大数量的Epoch管理器

[0188] Int epochQuorum;//EM群体大小

[0189] Int oldQuorum;//以前的EM群体大小

[0190] }

[0191] 每当EM主控的内容改变并且epochState!＝EpochStart时，EM主控在OFP中更新这个对象。所有的计算节点都订阅这个对象。

[0192] 这些字段具有以下语义：

[0193]

masterId	发布节点的节点 ID。该值由 OFP 提供。
masterEpoch	当前全局系统纪元值。
epochPreference	此纪元的偏好值。如果存在多个 EmMaster 对象，则所有节点都选择具有最高偏好值的节点。这用来在治疗裂脑情况时维护“最佳”分区纪元。
zkClientPort	用于客户端访问 Zookeeper 的 TCP 端口号。
zkServerPort	用于数据传输的 Zookeeper 服务器之间使用的 TCP 端口号。
zkElectionPort	用于领导选择的 Zookeeper 服务器之间使用的 TCP 端口号。
ofpDomain	要使用的 OFP 域 ID。
epochState	纪元的状态。可能的状态是 EpochStart、EpochInit、EpochRunning、EpochReconfig 和 EpochFail。
managers	被选为纪元管理器的节点的该组(节点 ID,随机数)元组。只有当可达、并且其 id 和随机数值都与其 EmNode 对象中的值相匹配时，才会将特定节点视为在管理器列表中。

[0194]

oldManagers	在上次重新配置事件发生时正在运行和可达的 EM 节点的该组（节点 ID，随机数）元组。这些节点的群体（由旧的群体定义）必须在重新配置期间保持可达，以避免故障。
maxManagerCount	预计最大数量的纪元管理器。
epochQuorum Epoch Manager	纪元管理器群体的大小。
oldQuorum	在上次重新配置事件时的纪元管理器群体的大小。

[0195] 所有计算节点状态信息(其也可以被称为“状态”)可以反映在发布的对象中,并且计算节点200存储很少的本地状态。换言之,在计算节点的EmNode对象(这是指代纪元管理器对象210的另一种方式)中反映用于计算节点200的内部状态,并且在EmMaster对象(其是指代纪元管理器主控对象212的另一种方式)中反映用于纪元管理器主控的内部状态。在一些情况下,SysEpochMan 16可以仅存储SysEpochMan 16最近发布的每个EmNode对象210的

最新版本的内部副本。EM主控可以使用发布的EmMaster对象的内容，因为其在主控权改变时在计算节点之间传输。

[0196] 在描述进程的元素时，更新命名对象字段应该被理解为更新内部副本，其中，如果对本地副本做出任何变化，则SysEpochMan然后在该进程结束时作为更新对象发布。而且，在本地产生的EmNode对象210中的字段被称为local.X，其中，X是字段名称。来自其他计算节点的EmNode对象中的字段被称为remote.X，其中，X是字段名称。EmMaster对象212中的字段被称为master.X，其中，X是字段名称。

[0197] 为各个计算节点定义该过程。然而，在一致行动的情况下，计算节点的集合可以间接定义全局行为。此外，该过程被定义为一组可能的事件，每个事件触发一个进程，并且每个事件都可能导致更新发布的对象。可能的事件是：

[0198]

对象事件（Object Event）	在该组 EmNode 和 EmMaster 对象中发生了变化，或者节点可达性已改变。

[0199]

纪元出现（Epoch Up）	节点上出现纪元（IPCB 18 功能正常）。
管理器出现（Manager Up)	节点已经作为纪元管理器（本地 IPCB 服务器功能正常）功能完善。
系统重新启动（System Restart）	节点内的代理已请求重新启动整个系统，放弃当前系统纪元。
节点关闭（Node Shutdown）	节点内的代理已经请求完全关闭节点。
IPCB 纪元（IPCB Epoch）	对于 IPCB 内的系统纪元值的监视开始。这提供了一种方式来确保相同的 IPCB 平面不绑定到两个系统纪元值。
主控权改变（Mastership Change）	节点的硬件主控权状态已改变（对于具有这种硬件的节点）。

[0200] 在计算节点200发布纪元管理器对象210并确定计算节点200是纪元管理器主控(在上述假设下)之后，SysEpochMan 16可等待至少具有纪纪元管理器能力的计算节点的群体，来发布纪元管理器对象210。作为一个示例，SysEpochMan 16可以通过以下方式确定纪元管理器群体的大小：

[0201] 如果master.maxManagerCount>＝3或master.maxManagerCount＝＝1，则群体是

master.epochQuorum节点。

[0202] 如果master.maxManagerCount＝＝2,则群体是master.epochQuorum节点,其中,一个节点报告remote.hwMaster＝＝True(或者相反,没有remote.hwMaster＝＝True的单个节点不是群体)。

[0203] 一旦到达群体,SysEpochMan 16可以发布具有设置为启动新纪元的“EpochInit”的epochState字段的纪元管理器主控对象212。SysEpochMan 16可以执行纪元管理器主控初始化进程,以如下所述初始化纪元管理器主控对象212的状态。

[0204] masterId 节点ID(由OFP提供)。

[0205]

masterEpoch	master.masterEpoch 的旧值，或如果没有旧的 EmMaster 对象，则为一个随机数。
epochPreference	如果存在旧的 EmMaster 对象，则为 master.epochPreference 的旧值。通常来说，如果存在不止一个，则这个值应该表示分区的“良好”，以便存在“最佳”分区；这可能应基于 OFP 域 X 中的对象的数量或者可能基于可达的 OFP 节点的数量或其某种组合。
zkClientPort	master.zkClientPort 的旧值，或者如果没有旧的 EmMaster 对象，则为如下所述导出的值。
zkServerPort	master.zkServerPort 的旧值，或者如果没有旧的 EmMaster 对象，则为如下所述导出的值。
zkElectionPort	master.zkElectionPort 的旧值，或者如果没有旧的 EmMaster 对象，则为如下所述导出的值。
ofpDomain	master.ofpDomain 的旧值，或者如果没有旧的 EmMaster 对象，则为如下所述导出的值。
epochState	master.epochState 的旧值，或者如果没有旧 EmMaster 对象，则为 EpochStart。
managers	master.managers 的旧值，或者如果没有旧的 EmMaster 对象，则为空集。
oldManagers	master.oldManagers 的旧值，或者如果没有旧 EmMaster 对象，则为空集。
maxManagerCount	master.maxManagerCount 的旧值。如果没有旧的 EmMaster 对象，则使用任何 remote.maxEmCapableNodes 的最大值。如果该值为 0，则使用 3。
epochQuorum	master.epochQuorum 的旧值。如果没有旧的 EmMaster 对象，则使用值

[0206]

	（master.maxManagerCount/2）+1。（对于 hw 主控节点，XXX 应该为 1）
oldQuorum	master.oldQuorum 的旧值，或者如果没有旧的 EmMaster 对象，则为 0。

[0207] 纪元管理器主控计算节点接下来可以更新纪元管理器主控状态。

[0208] 更新状态的本地副本,但是只有当明确地提到时,该状态才被写回到OFP中,作为更新的对象。EM主控计算节点如下所述更新EmMaster状态:

[0209] |更新master.epochPreference的值。注意,这个值不应该经常改变,并且特别是在对象事件进程的每个周期都不能改变,否则其将永远不会会聚(每次更新都会触发新的对象事件)。关于这个如何设置的讨论,见部分8.4.12。

[0210] (查看是否有计算节点正在请求重新启动。)

[0211] |如果任何remote.restartRequest!=None并且remote.epoch==master.masterEpoch或remote.epoch==0:(某个节点正在请求重新启动)

[0212] ||设置master.epochState=EpochFail。

[0213] ||如果remote.restartRequest==ResetManagers,设置master.managers=<empty set>。

[0214] ||更新EmMaster对象并退出对象事件进程。

[0215] (管理纪元管理器组。)

[0216] |打开master.epochState:

[0217] ||情况EpochStart:

[0218] |||如果对于其经由OFP可达remote.emCapable==True的节点的群体(参见部分8.4.10):(组成群体的足够节点)

[0219] ||||设置master.epochState=EpochInit。

[0220] ||情况EpochInit:

[0221] |||根据部分8.4.14更新纪元管理器组。

[0222] |||如果少于对于其经由OFP可达remote.emCapable==True的节点的群体(参见部分8.4.10),则设置master.epochState=EpochFail。

[0223] |||如果master.managers中的节点的群体(见部分8.4.10)经由OFP可达并且均报告remote.managerUp和remote.epoch==master.masterEpoch,则设置master.epochState=EpochRunning。

[0224] ||情况EpochRunning:

[0225] |||如果少于master.managers中的节点的群体(见部分8.4.10)经由OFP可达:

[0226] ||||设置master.epochState=EpochFail。

[0227] |||否则:(群体是可达的)

[0228] ||||根据部分8.4.14更新纪元管理器组。如果master.managers改变,则设置master.epochState=EpochReconfig。

[0229] ||情况EpochReconfig:

[0230] |||如果少于Master.managers中的节点的群体(见部分8.4.10)经由OFP可达:(新的EM组已丢失群体)

[0231] ||||设置master.epochState=EpochFail。

[0232] |||否则,如果由master.oldManagers中的节点的master.oldQuorum(见部分8.4.10)定义的群体经由OFP可达:(旧的EM组已丢失群体)

[0233] ||||设置master.epochState=EpochFail。

[0234] |||否则:(群体是可达的)

[0235] ||||如果master.managers中的节点的群体(见部分8.4.10)经由OFP可达并且均报告remote.managerUp和remote.epoch==master.masterEpoch,则设置master.epochState=EpochRunning。

[0236] 如果不存在旧的EmMaster对象,则SysEpochMan 16可以为系统纪元、IPCB端口和OFP域ID生成新的值。对于系统纪元,SysEpochMan 16可以从足够大(64位)的数字空间中选择一个随机数,以使得碰撞的概率不太可能(小于0.1%),并且为OFP域ID设置该值。

[0237] 然而,端口号空间要小得多,并且因而,SysEpochMan 16可以选择位于端口范围内的可被3整除的随机数,将该值分配给客户端端口,将该值+1分配给服务器端口,并将该值+2分配给选择端口。系统纪元写入到IPCB本身,并且如果不止一个纪元被绑定到IPCB平面,则将检测到该纪元,并且系统重新启动。新选择的纪元管理器主控发布IPCB写入"/SysEpochMan/SystemEpoch/<epoch>",其中,<epoch>是新系统纪元的字符表示。每个计算节点可以监听该路径上的变化,并且如果检测到冲突,则请求系统重新启动。

[0238] SysEpochMan 16接下来可以等待纪元管理器计算节点的群体来配置IPCB 18,形成IPCB总体,并且然后成功地执行IPCB 18,如通告布尔值设为真的remote.managerUp的纪元管理器计算节点的群体所指示的。当成功执行IPCB 18时,SysEpochMan 16可以将IPCB平面绑定到系统纪元,从而启动IPCB纪元事件。

[0239] 每个计算节点响应于该IPCB纪元事件执行IPCB纪元进程,这可能导致更新的对象被发布到OFP域0(在一些示例中总是如此)。该进程如下:

[0240] |在"/SysEpochMan/SystemEpoch/"路径上调用getChildren,请求新的监视。

[0241] |如果local.epoch!=0,并且存在具有与local.epoch不同的值的任何产物,

[0242] ||设置local.restartRequest=重新启动。

[0243] 在成功执行IPCB 18之后,SysEpochMan 16可以更新纪元管理器主控212的本地副本,以将epochState更新为"EpochRunning"的值,并发布纪元管理器主控212的更新的本地副本。此时,分布式操作系统10已经合并(或换言之是可操作的)并且可以支持执行专用功能,例如,SysMan20、分配器22、专用OFP域和应用。SysEpochMan 16可通过经由纪元管理器主控对象212的管理器字段指定一组新的纪元管理器计算节点,来随时改变这组纪元管理器计算节点,其中纪元管理器主控对象212的epochState字段设置为"EpochReconfig"。

[0244] 充当纪元管理器主控的SysEpochMan 16也可以维护该组纪元管理器计算节点。纪元管理器主控计算节点可以确保IPCB 18状态保持一致性,这意味着可能确保在纪元管理器组之间始终存在至少一个共同的计算节点。纪元管理器主控也可以维护系统中的纪元管理器的数量,随着该组计算节点的变化而增加和减少计数。

[0245] 纪元管理器管理进程的输入是该组可达的具有纪元管理器功能的计算节点和先

前纪元管理器主控状态。纪元管理器主控可以确保保留现有纪元管理器组，同时执行具有更高纪元管理器优先级的任何计算节点(其将从该组纪元管理器中去除具有较低纪元管理器优先级的计算节点)。为了满足IPCB动态重新配置的一致性要求，SysEpochMan 16可以确保在新旧纪元管理器组中至少有一个共同的计算节点。SysEpochMan 16可以在必要时在至少一个共同的计算节点规则下迭代形成中间纪元管理器组，直到形成新的纪元管理器组。

[0246] 该进程如下：

[0247] 首先计算master.maxManagerCount的新值：

[0248] |设置roughCount＝(可达的具有EM能力的节点数/3)|1

[0249] |设置minCount＝Min(master.maxManagerCount,3)

[0250] |设置master.maxManagerCount＝Min(Max(roughCount,minCount),7)

[0251] 在以上计算中，roughCount可以基于具有EM能力的节点的总数来表示纪元管理器(EM)计算节点的期望数量的粗略近似。低阶位被设置为保证是奇数(和非零)。接下来，minCount是EM节点的最低可能目标数量，是3和当前数量中较小的一个(以便适应一个和两个EM节点系统)。最后，SysEpochMan 16可以选择粗略计数和最小计数中的较大者，但是将两者中的较大者限制为7，因为额外的EM节点的值增加很小的值并且可能导致处理延迟。在一些示例中，没有主控权硬件的一个和两个EM节点系统可能以master.maxManagerCount＝＝1结束，具有主控权硬件的两个EM节点系统将总是以master.maxmanagerCount＝＝2结束，而所有其他系统将以在3至7的范围内的奇数结束。

[0252] 接下来，充当纪元管理器主控的SysEpochMan 16可以如下选择master.managers的新值：

[0253] |将预期的管理器组设置为master.managers的所有OFP可达成员，报告remote.managerUp＝＝True。如果具有超过master.maxManagerCount的成员，则丢弃具有remote.emPriority的最低值的足够成员，以修剪该组的大小。

[0254] |添加具有remote.epoch＝＝master.masterEpoch的所有可达的具有EM能力的节点，其中该节点的remote.emPriority的值大于预期的管理器组的任何成员，在必要时用remote.emPriority的最低值替换现有成员，以便保持组的大小小于或等于master.maxManagerCount。

[0255] |如果预期的管理器组不包括master.managers中的任何的节点，报告remote.managerUp＝＝True，将具有remote.emPriority的最低值的预期的成员替换为具有remote.managerUp＝＝True和remote.emPriority的最高值的可达的当前成员。

[0256] |将master.oldManagers设置为master.managers。

[0257] |将master.managers设置为预期的管理器组。

[0258] |将master.oldQuorum设置为master.epochQuorum。

[0259] |将master.epochQuorum设置为(master.maxManagerCount/2)+1。

[0260] 上述进程的一个效果是保持管理器组稳定，同时当到达时，可能支持较高优先级的具有EM能力的节点。如果所产生的组不与当前组重叠，则SysEpochMan 16可以从当前组中选择一个(因为IPCB服务器组在某些情况下必须总是重叠，以保留共享文件系统)。对于一个和两个EM系统，master.epochQuorum的值可以设置为1，并且对于较大的系统，值将至少为2。

[0261] 支持分布式操作系统10的执行的每个节点(包括计算节点200)还可以单独监控参与该群体的纪元管理器节点的健康状况。当节点还没有确定纪元正在运行时或者当由于群体还不稳定而纪元处于“EpochReconfig”状态时,不会发生检测到群体的故障。检测群体的故障的进程如下:

[0262] |如果没有EmMaster对象,则以无故障退出。

[0263] |如果local.epochUp==False,则以无故障退出。

[0264] |如果master.epochState==EpochReconfig,则以无故障退出。

[0265] |通过计数remote.epoch==master.masterEpoch和remote.managerUp==True中的所有可达的EmNode对象,确定可达的纪元内EM节点的数量。

[0266] |如果节点数小于master.epochQuorum,则以故障退出。

[0267] |如果节点数为1,且master.maxManagerCount==2且local.hwMaster==False,则以故障退出。(网络已分区,并且本地节点不是硬件主控。)

[0268] |否则,以无故障退出。

[0269] 当群体发生故障时,监控计算节点将检测到故障并重新启动。群体验证可以取决于经由纪元管理器主控对象212发布的纪元状态。例如,在状态“EpochInit”中,该组纪元管理器正在会聚,因此,在该组OFP可达纪元管理器计算节点之中,通过群体的丢失检测到故障。在状态“EpochRunning”中,计算节点可以在报告remote.managerUp的该组纪元管理器计算节点之中,通过群体的丢失检测到故障。在状态“EpochReconfig”中,新的一组纪元管理器仍然在会聚,并且当在该组纪元管理器节点之中发生群体的丢失时,计算节点检测到故障。

[0270] 鉴于上述情况,假设在启动之前存在纪元,初始启动有一些考虑因素。在初始启动期间,没有节点维护纪元管理器对象210或纪元管理器主控对象212。因而,所有节点异步启动并发布初始纪元管理器对象212,其中,第一具有纪元管理器功能的节点将其自身选择为纪元管理器主控。其他节点随后跟随自选的纪元管理器主控节点。此时,由于尚未创建纪元,每个节点的local.epoch等于零。

[0271] 自选的纪元管理器主控保持在“EpochStart”状态中,并且不将任何更新发布到纪元管理器主控对象212的EmMaster字段,直到具有纪元管理器功能的节点的群体经由纪元管理器对象210通告自己并一致同意纪元管理器主控节点的身份。在一些情况下(例如,竞赛条件),不止一个节点可以选择自己作为纪元管理器主控。该进程可能不会前进,直到主控权会聚。

[0272] 假设主控权会聚,计算节点200将其自身选择为纪元管理器主控,则计算节点200的SysEpochMan 16以新系统纪元(例如,值1)、新的纪元管理器组以及EpochInit状态发布纪元管理器主控对象212。然后,所有节点可以用新纪元更新其相应的纪元管理器对象210,公布更新的纪元管理器对象210,以确认新纪元的初始化。新选择的纪元管理器节点执行IPCB18。

[0273] 接下来,所有节点发布写入到IPCB 18(具有充当IPCB服务器的全组纪元管理器),以便检测IPCB 18的成功执行。所有纪元管理器节点也可以写入IPCB 18(其本身作为唯一服务器),以在加入IPCB总体时检测成功。当IPCB总体会聚时,待完成的IPCB写入将完成,导致所有节点经由纪元管理器对象210发布“epochUp”的epochStatus,并且纪元管理器节点

发布“managerUp”状态。

[0274] 当epochUp事件发生时(意味着当epochState指示“epochUp”的值时),每个节点执行纪元出现进程。由在上述对象事件进程中到IPCB 18内的写入的异步完成来触发epochUp事件,所有EM节点作为服务器,表示已成功形成纪元的IPCB群体。如通常那样,如果该进程修改任何对象,则在OFP中更新。该进程如下:

[0275] 设置local.epochUp=True。

[0276] 一旦纪元管理器的群体达到managerUp状态,则所有节点执行以下管理器出现进程。当发生managerUp事件时,选为纪元管理器的每个节点都可以执行管理器出现进程。由在对象事件进程中到IPCB 18内的写入的异步完成来触发该事件,而只有本地节点作为服务器,表示本地节点已成功加入IPCB群体。如果此进程修改任何对象,则在OFP中更新该对象。该进程如下:

[0277] 设置local.managerUp=True。

[0278] 在执行管理器出现进程之后,纪元管理器主控节点发布更新的纪元管理器主控对象212,epochState设置为“EpochRunning”的值。响应于“EpochRunning”的更新的epochState,每个节点然后执行软件栈11的上层。

[0279] 此外,OFP 14(由数字逻辑电路中实现的一个或多个处理器执行,在图3中未示出)包括状态208。状态208包括数据结构,诸如,存储多个对象的树数据结构(例如,基数树),每个对象定义用于分布式操作系统10和在由分布式操作系统10提供的应用空间中执行的应用中的至少一者的状态信息的一部分。

[0280] 通常,计算节点200(更具体地,一个或多个处理器)形成用于状态208的树数据结构,以包括多个分层排列的树节点,每个树节点存储用于消息片段的数据,包括例如摘要和片段ID。对于状态208的树数据结构可以根据树节点的片段ID进行排列。

[0281] 计算节点200可以被配置为使状态208与执行分布式操作系统10的其他实例的其他节点的对应数据结构同步。通常,当执行分布式操作系统10的实例的每个节点的树数据结构具有每个树数据结构内的树节点的共同排列和互连时,并且当树数据结构的对应树节点具有相同摘要值时,状态208可以被认为与执行分布式操作系统10的其他实例的其他节点的数据结构同步。

[0282] 计算节点200可进一步确定状态208是否与执行分布式操作系统10的其他实例的其他节点的状态数据结构同步。计算节点200可递归地行进表示状态208的树数据结构的树节点。如果表示状态208的树数据结构的树节点的本地摘要与执行分布式操作系统10的其他实例的节点的树数据结构的对应树节点的摘要匹配,则计算节点200可以确定树节点是同步的。否则,计算节点200可以确定对状态208的更新是必要的。

[0283] 如果更新是必要的,则计算节点200可以确定其状态208的版本是否是最新的,或者存储执行分布式操作系统10的另一实例的不同节点的状态信息的树数据结构的另一版本是否是最新的。如果表示计算节点200的状态208的树数据结构的树节点的版本是最新的(例如,具有最新的逻辑时钟值),则计算节点200可以将表示状态208的树数据结构的树节点的消息片段数据发送到执行分布式操作系统10的其他实例的一个或多个其他节点。否则,如果表示计算节点200的状态208的树数据结构的树节点的版本不是最新的,则计算节点200可以从执行最新的分布式操作系统10的另一实例的其他节点中的一个不同节点接收

表示状态208的树数据结构的树节点的消息片段数据，并且用接收到的消息片段数据更新表示状态208的树数据结构的树节点。

[0284] 图4A至图8B是示出图2所示的多机箱路由器内的节点操作，在解决可能影响根据本公开中描述的技术维护的分布式操作系统的执行的各种情况时的各个方面的框图。图4A至图4D是示出根据本公开中所描述的技术的各个方面的用于在分布式操作系统的执行期间解决纪元管理器故障时的节点302A-302F(“节点302”)的操作的框图。每个节点302可以基本上类似于图3中所示的计算节点200。

[0285] 在图4A的示例中，节点302可共同执行分布式操作系统300A，其中，节点302A作为纪元管理器主控(“EM主控”)操作，节点302D和302F作为选择的纪元管理器(“EM”)操作，节点302C作为未选择的纪元管理器操作但具有纪元管理器能力(“具有EM能力”)，并且节点302B和302E作为不具有EM能力来操作。使节点302互连的较粗加权线可以表示多播(或者在一些情况下为广播)链路，而使节点302互连的不太粗的加权线可以表示单播链路。

[0286] 分布式操作系统300A的状态是“epochRunning”，具有作为IPCB服务器执行的三个节点的群体(即，图4A的示例中的节点302A、302D和302F)。因而，只要形成群体的三个节点中的两个(鉴于(3/2)+1＝2)保持运行，就可以维护群体。换言之，尽管形成群体的节点302A、302D和302F中的一个节点故障，但分布式操作系统300A仍然可以运行。

[0287] 在图4B的示例中，节点302A和302B发生故障，导致分布式操作系统300B。然而，由于两个纪元管理器(即，图4B的示例中的节点302D和302F)保持运行，所以分布式操作系统300B可以维护状态信息一致性并继续执行。鉴于纪元管理器主控302A发生故障，分布式操作系统300B将节点302D选为纪元管理器主控。

[0288] 鉴于群体只包括两个节点，群体不能失去另一纪元管理器节点，同时仍然维护分布式操作系统300B的操作。因而，纪元管理器主控302D可以使用上述进程来重新配置设置为包括节点302C的纪元管理器。在选择具有EM能力的节点302C作为纪元管理器进行操作时，节点302C可以作为IPBC服务器执行，并将所有的IPBC状态复制到由节点302C执行的本地IPBC服务器，从而导致分布式操作系统300C。

[0289] 在图4D的示例中，假设节点302A、302B和302D全部同时发生故障(即，在该示例中，从图4A的分布式操作系统300A转换到图4D所示的分布式操作系统300D)。丢失节点302A、302B和302D，分布式操作系统300D丢失纪元管理器的群体(和IPCB服务器)。

[0290] 因而，IPCB客户端可能已经写入IPCB服务器，并且接收到这种状态被写入到IPCB的确认，但是该状态仅在发生故障时存在于节点302A和302D处。在这种情况下，该状态是不可恢复的(或换言之，丢失)，并且因而，分布式操作系统300D发生故障，重新启动分布式操作系统300D的软件栈11中的一层或多层。当节点302重新启动时，节点302在系统纪元的新值上会聚，导致分布式操作系统300D，其中，节点302F充当纪元管理器主控并且节点302C充当纪元管理器。

[0291] 图5A至图5C是示出根据本公开中所描述的技术的各个方面的由于在分布式操作系统的执行期间的故障而解决节点302的分区时的节点302的操作的框图。每个节点302可以基本上类似于图3中所示的计算节点200。

[0292] 在图5A的示例中，由于链路312A-312C的故障，分布式操作系统310A已经分区，导致由节点302A和302B组成的第一分区和由节点302C-302F组成的第二分区。由于在第一分

区和第二分区之间没有通信，所以第一分区和第二分区中的每一个存在。

[0293] 从第二分区中的节点302C-302F的角度来看，节点302A和302B已经发生故障。在发生故障时，系统纪元值是42，并且第一分区和第二分区都以系统纪元值为42继续执行分布式操作系统310A。当发生故障时，第一分区的节点302A和302B确定纪元管理器的群体已经丢失，导致重新启动节点302A和302B的软件栈11的一层或多层。第二分区的节点302C-302F仅丢失单个纪元管理器(即，在该示例中，纪元管理器主控302A)，并且纪元管理器的群体不丢失，从而允许分布式操作系统310A继续由第二分区的节点302C-302F操作。

[0294] 图5B示出了由于链路312A-312C的故障而仍然不存在群体导致的不能重新启动和组织的第一分区的重新启动的结果。节点302A选择自身作为纪元管理器主控，但检测到群体的损失，并且因此不能在功能上参与分布式操作系统310B的执行(导致不能分配系统纪元值，因为第一分区不起作用，其中，在图5B的示例中，系统纪元值的缺失被表示为“??”)。

[0295] 节点302A存储纪元管理器组(例如，作为由OFP 14分配的节点ID的列表)和纪元管理器的群体中的节点的数量，通过重置进程维护纪元管理器组和群体中的节点的数量。因而，节点302A可以确定单个节点(即，图5B的示例中的节点302A)不足以满足(N/2)+1的群体阈值，其中，N是先前的群体中的节点的总数。即使在第一分区中存在足够数量的具有EM能力的节点时，节点302A也可以确定不满足群体阈值，这是因为来自先前群体的纪元管理器组与具有EM能力的节点的组不匹配。

[0296] 在第二分区中，选择节点302D作为纪元管理器主控，而节点302F保持为纪元管理器。节点302D可以使用上述进程来重新配置纪元管理器组，以包括节点302C，从而允许分布式操作系统310B即使在节点302C、302D和302F中的一个发生故障时也保持运行。

[0297] 在图5C的示例中，链路312A变成运行，允许分区合并(或换言之，“愈合”)。鉴于由于第一分区中未满足群体阈值而节点302A从不是运行的纪元管理器主控，一旦链路312A变为运行，节点302D保持为纪元管理器主控。节点302C和302F保持为纪元管理器，而节点302A降级为具有EM能力。因此，节点302执行系统纪元值为42的分布式操作系统310C。

[0298] 图6A和图6B是示出根据本公开中所描述的技术的各个方面的在分布式操作系统的执行期间在解决节点302的受控关闭时的节点302的操作的框图。再次，每个节点302可以与图3中所示的计算节点200基本相似。

[0299] 在一些情况下，系统管理器可能要求将一个或多个节点302从支持执行图5C所示的分布式操作系统310A中移除。在图6A的示例中，撤销(或换言之，移除)节点302D和302F，以支持图5C所示的分布式操作系统310A的执行，导致分布式操作系统320A。撤销节点302D导致纪元管理器主控丢失，而撤销节点302F导致纪元管理器丢失。此外，撤销三个纪元管理器中的两个(即，图6A的示例中的节点302D和302F)将导致群体的丢失。

[0300] 为了避免丢失群体，节点302D和302F在被撤销之前发出撤销相应的具有EM能力的状态的请求。节点302C可以接收请求，并且选择自身作为纪元管理器主控，使节点302A作为纪元管理器以维护群体(当两个节点满足(N/2)+1的群体阈值时)。在使节点302A作为纪元管理器时，节点302C可以将纪元管理器组重新配置以移除节点302D和302F，从而允许节点302D和302F撤销，并且从而不再支持分布式操作系统320A的操作，导致图6B所示的分布式操作系统320B。

[0301] 尽管示出为采取单次迭代来撤销一个或多个节点302，但可能存在需要多次迭代

来撤销一个或多个节点302的情况。只要至少一个节点纪元管理器仍然在新的一组纪元管理器与旧的一组纪元管理器之间，就可以撤销任何数量的节点。一个纪元管理器保持在旧的一组和新的一组纪元管理器之间的要求是保留IPCB状态。因此，在组中的所有纪元管理器都将被撤销的情况下，可以留下一个纪元管理器，形成中间的一组纪元管理器。一旦形成中间的一组纪元管理器，管理在旧的一组和中间的一组纪元管理器之间的转换的旧纪元管理器可以撤销，以形成新的一组纪元管理器。

[0302] 在单机箱路由器的一些示例中，仅包括一个或两个路由引擎以及一个或多个转发单元，作为一个示例，转发单元可以包括灵活的PIC集中器(FPC)。因为转发单元可能不能够运行IPCB 18，所以转发单元可能不具有EM能力。因而，系统在路由引擎中仅具有一个或两个具有EM能力的节点(并且因此仅具有一个或两个IPCB服务器)。

[0303] IPCB动态重新配置机制的要求，即旧的和新的总体成员资格通过至少一个节点重叠，这本质上意味着必须有至少三个IPCB节点，使该进程有用(离开的这个人、周围的这个人和新人)。此外，IPCB可能至少需要两个节点来运行，因为这是最小的群体。

[0304] 为了在具有EM能力的一或两节点系统中执行，IPCB可以以不同的方式在该具有EM能力的一或两节点系统上操作。修改IPCB可能存在两个问题：如何保持IPCB运行和一致性，以及如何在具有两个EM能力的节点系统分区时避免裂脑。

[0305] IPCB可以以复制和独立两种模式执行。在复制模式下，存在多个IPCB服务器并且它们彼此协调(并且必须至少有两个)。在独立模式下，只有一个IPCB服务器。IPCB可能会重新启动，以在模式之间切换。

[0306] 在单节点系统上，IPCB可以在独立模式下运行。在这个系统中没有冗余，所以在本文描述的方案中没有特别的问题——如果唯一的纪元管理器发生故障，则系统也发生故障。

[0307] 在具有EM能力的两节点系统上，当一个具有EM能力的节点发生故障并然后恢复时，IPCB可以在复制模式和独立模式之间来回切换。当从独立模式进入复制模式时，一致性可以得到保证，这是因为只有从第一节点上的本地文件系统上的IPCB事务日志中重新加载的IPCB状态的一个副本，而第二IPCB服务器从第一IPCB服务器接收所有状态。从复制模式到独立模式时，一致性可能会得到保证，因为具有EM能力的两节点系统的IPC群体尺寸是2，这可能会导致在提交事务之前将最新事务写入其事务日志的节点。在重新启动之后留下的单个IPCB服务器可以以这种方式具有所有事务。

[0308] 当具有EM能力的两节点系统变为分区时，可能发生下面更详细地描述的裂脑情况(其中，双方将出现在独立模式中)。然而，两具有EM能力的系统具有主控权硬件，即指定一个节点或另一节点为主控的FPGA。可以利用主控权硬件，并且可以在具有EM能力的两节点系统中调整群体规则，以定义存在群体的时间仅仅是一个节点可达的时间并且该节点具有硬件主控权。该群体规则调整解决了裂脑情况，这是因为两个节点中只有一个节点将是主控，并且另一节点将没有群体，并且因此将重新启动并保持关闭，直到分区合并为止。

[0309] 图7A至图7C是示出根据本公开中所描述的技术的各个方面的由于分布式操作系统的执行期间的故障而导致在解决节点302的多个分区时的节点302的操作的框图。每个节点302可以基本上类似于图3中所示的计算节点200。

[0310] 在图7A的示例中，链路312A-312E全部发生故障，导致三个分区。第一分区包括节

点302A和302B。节点302A将其自身选择为纪元管理器主控，但由于在第一分区中设置的旧纪元管理器的纪元管理器不足以满足(N/2)+1的群体阈值，因此不能重新建立群体。

[0311] 第二分区包括节点302C和302D。在链路312A-312E发生故障之前，节点302D是纪元管理器。节点302D可以选择自身作为纪元管理器主控，但由于来自先前群体的三个纪元管理器节点中的两个(即，图7A的示例中的节点302A和302F)不可用，所以不能维护群体。

[0312] 第三分区包括节点302E和302F。在链路312A-312E发生故障之前，节点302F是纪元管理器。节点302F可以选择自身作为纪元管理器主控，但由于来自先前群体的三个纪元管理器节点中的两个(即，图7A的示例中的节点302A和302D)不可用，所以不能维护群体。

[0313] 因而，产生图7B中所示的分布式操作系统330B，其中，三个分区中没有一个能够执行分布式操作系统330B。因为节点302都不能执行分布式操作系统330B，所以每个分区的系统纪元值是未知的(在图7B中，由“??”表示)。每个分区的纪元管理器主控(即，在图7B的示例中，节点302A、302D和302F)等待，直到链路312A-312E中的一个或多个变为运行的，以重新形成群体并继续执行分布式操作系统。

[0314] 在图7C的示例中，由于链路312A和312E变为运行的，所以分区已经合并。节点302协商先前的纪元管理器主控中的哪个将保持为主控(例如，通过上面讨论的EM主控优先级)。在图7C的示例中，节点302F保持为纪元管理器主控，而节点302D和302A作为纪元管理器执行。因而，节点302交换状态信息，以重新获得一致性，并且更新到系统纪元值43(如图7A的示例中所示，从42)。节点302可共同执行系统纪元值为43(以与由系统纪元值42标识的版本区分开)的分布式操作系统330C。

[0315] 图8A和图8B是示出根据本公开中所描述的技术的各个方面的由于分布式操作系统的执行期间的故障而导致的在解决“裂脑”情况时的节点302的操作的框图。每个节点302可以基本上类似于图3中所示的计算节点200。

[0316] 裂脑情况是指系统分为两个或更多个分区的情况，其中，由于不知道其它分区仍然运行因此至少两个分区保持运行，导致分离或分裂的执行环境(或换言之，“大脑”)。在分布式操作系统先前已经执行的正常操作中，如上所述，由群体阈值和先前的一组纪元管理器调节的群体系统避免了裂脑的情况。

[0317] 当系统/装置在没有先前状态(例如，具体而言，没有设置群体大小和/或阈值)的情况下启动并且装置的节点分区时，可能发生裂脑情况。在图8A的示例中，由于链路故障，节点302划分为两个分区，其中，第一分区包括节点302A和302B，并且第二分区包括节点302C-302F。节点302A被选为第一分区的纪元管理器主控，并将节点302B指定为第一分区的纪元管理器。节点302D被选为第二分区的纪元管理器主控，并将节点302C和302F中的每一个指定为纪元管理器。

[0318] 在这种裂脑情况下，分布式操作系统340A的第一分区可以选择1234的系统纪元值，而分布式操作系统340A的第二分区选择42的系统纪元值。考虑到系统纪元值表示分布式操作系统340A的版本并且允许在不同版本的分布式操作系统340A之间的适当同步，在分布式操作系统340A的初始启动期间对系统纪元值的选择是随机的，以避免两个分区选择相同的系统纪元值，因为这会影响不同节点之间的同步。

[0319] 假设一个链路变成如图8B的示例中所示的操作，则分布式操作系统340B的两个分区合并。SysEpochMan 16使用偏好机制(如上所述)来确定要保留哪个纪元以及要丢弃哪个

纪元,以避免在分区合并之后重新启动最新(或“最佳”)版本。

[0320] 图9是示出图3所示的多机箱路由器的节点在执行本公开中描述的分布式操作系统技术的各个方面时的示例性操作的流程图。如上所述,计算节点200首先执行OFP 14,以确定允许由分布式操作系统10的单个实例的节点合并和执行的节点的拓扑(400)。OFP物理拓扑发现可以以类似于链路状态协议的方式发生。OFP 14基于通告构建表示通过链路彼此互连的主要节点和次要节点的拓扑的图形数据结构。

[0321] 接下来,计算节点200可以执行SysEpochMan 16,SysEpochMan 16可以基于表示主要节点和次要节点的拓扑的图形数据结构从被配置为作为纪元管理器执行的那些节点之中选择纪元管理器主控(402)。所选的纪元管理器主控可以选择一个或多个纪元管理器(包括所选的纪元管理器主控)充当纪元管理器(404)。然后,每个纪元管理器可以执行IPCB 18(404)。

[0322] IPCB 18形成服务器和客户端的网络。这些服务器可以被称为ICPB总体。IPCB 18可以建立纪元管理器的群体,其中,大多数服务器(例如,大于(N/2)+1,其中,N表示服务器/纪元管理器的数量)连接并且用于IPCB 18,以继续分布式操作系统10的成功操作(408)。以这种方式,为了执行分布式操作系统10的目的,这些技术可以允许分离的(或换言之,单独的)计算节点合并。

[0323] 在形成群体并建立通过其客户端可与共享文件系统交互的IPCB 18之后,IPCB 18可监控IPCB服务器(这是表示纪元管理器的另一种方式),以检测纪元管理器故障(例如,通过多个节点中的一个或多个节点之间的连接是否发生故障来测量)(410)。当没有发生连接故障时(“否”412),IPCB 18继续监控群体,以检测纪元管理器故障(410)。

[0324] 当IPCB纪元管理器发生故障或链路发生故障时(其通常可被称为“连接故障”)(“是”412),剩余的IPCB纪元管理器可以确定纪元管理器的群体是否存在。剩余的IPCB纪元管理器可以通过比较运行的纪元管理器(由变量“N”表示)的数量是小于群体阈值(例如(N/2)+1)来确定纪元管理器的群体是否存在(414)。

[0325] 当运行的纪元管理器的数量小于群体阈值(“是”414)时,剩余的纪元管理器可以重新启动分布式操作系统10(其可以不需要重新启动多机箱路由器4或者内核12,而是仅重新启动软件栈11中的内核12之上的那些层中的一个或者多个,诸如,OFP 14、SysEpochMan 16、IPCB 18、SysMan 20和/或分配器22)(416)。在重新启动时,该进程再次启动,执行协议,以确定节点的拓扑等(400-410)。当运行的纪元管理器的数量大于或等于群体阈值(“否”414)时,剩余的纪元管理器可以维护群体并继续操作(监控群体,以检测纪元管理器故障410),可能向群体增加在形成群体期间未被选为纪元管理器的新纪元管理器。

[0326] 图10是示出根据本公开的技术的用于存储状态信息的示例树数据结构470的概念图。在这个示例中,树数据结构470包括根树节点450和树节点452-464。每个树节点452-464包括前缀值和摘要值。在这个示例中,树节点452的前缀值可以是XX/104,树节点454的前缀值可以是XXX/108,树节点456的前缀值可以是XXY/108,树节点458的前缀值可以是XXXX/112,树节点460的前缀值可以是XXXY/112,树节点462的前缀值可以是XXYX/112,并且树节点464的前缀值可以是XXYY/112。在这个示例中,树节点458-464是树数据结构470的叶树节点,因为树节点458-464没有任何子树节点。

[0327] 树数据结构470的每个树节点还包括摘要值。通常,每个摘要值表示其所表示的块

体中的所有片段。因此，根450包括表示树数据结构470中的所有消息的摘要。树节点454的摘要覆盖树节点454、458和460的消息片段，而树节点456的摘要覆盖树节点464的消息片段，为了确定两个树数据结构(诸如，树数据结构470)是否相同，可以比较树节点452的摘要和与树数据结构470进行比较的不同树数据结构的对应树节点的摘要，并且如果这些摘要中的每一个在树数据结构之间匹配，则可以说树数据结构是相同的，并且因此是同步的。

[0328] 如果两个这种树数据结构不同步，则诸如图3的计算节点200等节点可以递归地行进树数据结构470，以确定树节点452-464中的哪个要更新。计算节点200可以在树节点452处开始并且将树数据结构470向下行进到叶树节点，即树节点458-464。计算节点200然后可将每个叶树节点458-464与另一树数据结构的对应叶树节点进行比较。对于不匹配的每个叶树节点，计算节点200可以与正在存储另一树数据结构的分布式操作系统10的另一节点交换消息，以如上所述同步树数据结构的对应树节点。

[0329] 图11是示出根据本公开的技术的用于在分布式操作系统的由网络装置的相应计算节点执行的不同实例之间同步状态信息的示例性方法的流程图。尽管应该理解附加的节点可以执行基本相似的方法，但是在这个示例中，描述了两个节点。每个节点可以包括与相对于图3的计算节点200所讨论的组件类似的组件。

[0330] 在该示例中，第一节点首先构建包括多个对象的数据结构，每个对象存储状态信息(500)。数据结构可以是如上所述的树数据结构。因此，树数据结构的构建可以进一步涉及计算树数据结构的叶树节点的摘要以及树数据结构的非叶树节点的摘要。非叶树节点的摘要可以表示对应的树节点和对应的树节点可访问的树节点(例如，子树节点向下到叶树节点)的数据。状态信息可以是例如用于分布式操作系统本身和/或在由分布式操作系统提供的应用空间中执行的一个或多个应用的状态信息。对象可以表示消息或消息片段，如上所述。此外，对象也可以根据对象洪泛协议(OFP)来分配，也如上所述。因此，例如，第一节点根据OFP将对象洪泛到网络装置的其他计算节点(502)。因此，在本示例中的第二节点接收对象(504)并存储包括对象的数据结构(506)。

[0331] 随后，第一节点接收更新的状态信息(508)。例如，第一节点可以接收用于一个应用或用于分布式操作系统的更新的状态信息。作为响应，第一节点更新数据结构的相关对象(510)，即对应于更新的状态信息的对象，以存储更新的状态信息。当更新数据结构的对象时，第一节点也可以更新与数据结构的对象相关联的逻辑时钟值，以表示更新数据结构的对象的时间。如上所述，假设数据结构是树数据结构，则第一节点可以更新对应于更新的状态信息的树数据结构的树节点以及在根树节点与受更新的状态信息影响的层次最低的树节点之间的树数据结构的每个树节点的摘要。

[0332] 而且，在更新数据结构之后，第一节点例如根据OFP将更新的对象(消息或消息片段)洪泛到网络装置的其他计算节点(512)。第一节点还使用更新的数据结构更新其配置(514)。例如，假设数据结构是树数据结构，网络装置的第一计算节点和第二计算节点可以比较树数据结构的对应树节点的摘要，以确定树数据结构的对应树节点是否匹配(即，具有相同的摘要)。对于不具有匹配的摘要的树数据结构的每个树节点，第一节点(在本示例中，假设具有更新版本的状态信息)将对象数据(即，树节点的消息片段)洪泛到第二节点。更通常地，网络装置的第一计算节点和第二计算节点(以及网络装置的任何其他计算节点)可以比较树数据结构的对应树节点的逻辑时钟值，以确定哪个树数据结构具有树数据结构的最

新版本的树节点，并且然后，具有树数据结构的最新树节点的网络装置的计算节点将树数据结构的树节点的对象洪泛到网络装置的其他计算节点。

[0333] 响应于从第一节点接收到洪泛的对象(516)，第二节点还以类似于第一节点的方式更新其数据结构的对象(518)，并还使用更新的数据结构更新其配置(520)。

[0334] 以这种方式，图11的方法表示一种方法的示例，该方法包括：由网络装置的执行分布式操作系统的第一实例的第一计算节点，接收用于分布式操作系统和在由分布式操作系统提供的应用空间中执行的应用中的至少一者的更新的状态信息；由网络装置的第一计算节点更新网络装置的第一计算节点的本地数据结构，以包括更新的状态信息，本地数据结构存储多个对象，每个对象定义用于分布式操作系统和应用中的至少一者的状态信息的一部分；以及由网络装置的第一计算节点，使更新的本地数据结构与由网络装置的第二计算节点执行的分布式操作系统的第二实例的远程数据结构同步。

[0335] 尽管在图11中，只有网络装置的第一计算节点被示出为接收更新的状态信息，但是应当理解的是，在其他示例中，网络装置的其他计算节点可以接收更新的状态信息并将对应的对象洪泛到第一节点。例如，相对于图11讨论的第二节点可以接收更新的状态信息，更新其数据结构以包括更新的状态信息，然后将表示更新的状态信息的对象洪泛到网络装置的第一计算节点。如上所述，通常，网络装置的每个计算节点比较相应树数据结构的树节点的摘要，以确定树数据结构的树节点是否匹配。当树数据结构的对应树节点的摘要不匹配时，网络装置的计算节点可以比较与树数据结构的树节点相关联的逻辑时钟值，以确定哪个树数据结构包括树数据结构的最新的树节点。具有树数据结构的最新的树节点的网络装置的计算节点将用于树数据结构的树节点的数据洪泛到网络装置的其他计算节点。

[0336] 本文描述的一种或多种技术可以部分或全部以软件执行。例如，计算机可读介质可以存储或以其他方式包括计算机可读指令，即，可以由处理器执行的程序代码，以执行上述技术中的一个或多个。例如，计算机可读介质可以包括随机存取存储器(RAM)、只读存储器(ROM)、非易失性随机存取存储器(NVRAM)、电可擦除可编程只读存储器(EEPROM)、闪存、磁或光学介质等。

[0337] 已经描述了本公开的各种实施方式。尽管参考多机箱路由器(每个机箱包括多个路由引擎)进行了描述，但是这些技术可以应用于在至少一个机箱中具有多个控制节点的任何多机箱装置。其他装置的示例通常包括交换机、网关、智能集线器、防火墙、工作站、文件服务器、数据库服务器和计算装置。此外，所描述的实施方式涉及分级排序和时间链接的数据结构，但是其他实施方式可以使用不同的数据结构。

[0338] 除了或者代替上述内容，还描述下面的示例。在任何以下示例中描述的特征可以与本文描述的任何其他示例一起使用。

[0339] 实施例1.一种网络装置包括多个硬件计算节点，硬件计算节点被配置为执行分布式操作系统，多个硬件计算节点中的至少一个被配置为：确定多个硬件计算节点中的一个或多个是否发生故障并且不再支持分布式操作系统的执行；确定多个硬件计算节点中的剩余硬件计算节点是否超过群体(quorum)阈值，并且当多个硬件计算节点中的剩余硬件计算节点小于群体阈值时，重新启动分布式操作系统。

[0340] 实施例2.根据实施例1所述的网络装置，其中，分布式操作系统进一步被配置为：设置纪元值以表示分布式操作系统的当前版本；并且将纪元值加1，使得分布式操作系统的

实例避免执行分布式操作系统的两个不同版本。

[0341] 实施例3.根据实施例1所述的网络装置，其中，分布式操作系统进一步被配置为：标识被配置为会聚在分布式操作系统的单个版本上的多个硬件计算节点的群体；将多个硬件计算节点的群体中的一个硬件计算节点选择为纪元管理器主控硬件计算节点，其中，该纪元管理器主控硬件计算节点确定多个硬件计算节点的群体内的连接是否已出现故障。

[0342] 实施例4.根据实施例3所述的网络装置，其中，群体阈值等于硬件计算节点的群体中的数量除以2加1((N/2)+1)，其中，N表示多个硬件计算节点的群体中的数量。

[0343] 实施例5.根据实施例3所述的网络装置，其中，参与群体的多个硬件计算节点中的一个被配置为：确定参与群体的多个硬件计算节点中的一个已丧失到该群体的连接，并且响应于多个硬件计算节点中的一个已丧失到群体的连接的确定，重新启动参与群体的多个硬件计算节点中的一个。

[0344] 实施例6.根据实施例3所述的网络装置，其中，多个硬件计算节点被配置为在连接故障不超过连接故障阈值时执行当前版本的分布式操作系统。

[0345] 实施例7.根据实施例1所述的网络装置，其中，该多个硬件计算节点进一步被配置为：执行协议，通过该协议来发现多个硬件计算节点的拓扑；基于拓扑确定多个硬件计算节点的子集，以管理分布式操作系统的执行；执行通信总线，通过该通信总线在多个硬件计算节点的子集之间同步操作系统状态信息；以及基于操作系统状态信息执行分布式操作系统，以提供其中一个或多个应用执行的执行环境。

[0346] 实施例8.一种方法，包括：由网络装置内包括的多个硬件计算节点中的至少一个确定多个硬件计算节点中的一个或多个是否发生故障；由多个硬件计算节点中的至少一个确定多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值；以及当多个硬件计算节点中的剩余硬件计算节点小于群体阈值时，由多个硬件计算节点中的至少一个重新启动分布式操作系统。

[0347] 实施例9.根据实施例8所述的方法，进一步包括：设置纪元值以表示分布式操作系统的当前版本；并且将纪元值加1，使得分布式操作系统的实例避免执行两个不同版本的分布式操作系统。

[0348] 实施例10.根据实施例8所述的方法，进一步包括：标识被配置为会聚在分布式操作系统的单个版本上的多个硬件计算节点的群体；将多个硬件计算节点的群体中的一个硬件计算节点选择为纪元管理器主控硬件计算节点；以及由该纪元管理器主控硬件计算节点确定多个硬件计算节点的群体内的连接是否已出现故障。

[0349] 实施例11.根据实施例10所述的方法，其中，群体阈值等于多个硬件计算节点的群体中的数量除以2加1((N/2)+1)，其中，N表示多个硬件计算节点的群体中的数量。

[0350] 实施例12.根据实施例10所述的方法，进一步包括：由参与群体的多个硬件计算节点中的一个确定参与群体的多个硬件计算节点中的一个已丧失到该群体的连接；并且响应于多个硬件计算节点中的一个已丧失到群体的连接的确定，由参与群体的多个硬件计算节点中的一个重新启动参与群体的多个硬件计算节点中的一个。

[0351] 实施例13.根据实施例10所述的方法，进一步包括：在连接故障不超过连接故障阈值时执行当前版本的分布式操作系统。

[0352] 实施例14.根据实施例8所述的方法，进一步包括：执行协议，通过该协议来发现多

个硬件计算节点的拓扑;基于拓扑确定多个硬件计算节点的子集,以管理分布式操作系统的执行;执行通信总线,通过该通信总线在多个硬件计算节点的子集之间同步操作系统状态信息;以及基于操作系统状态信息执行分布式操作系统,以提供其中一个或多个应用执行的执行环境。

[0353] 实施例15.一种其上存储有指令的非暂时性计算机可读存储介质,该指令在执行时使网络装置的一个或多个处理器配置为:确定运行分布式操作系统的多个硬件计算节点中的一个或多个是否发生故障;确定多个硬件计算节点中的剩余硬件计算节点是否超过群体阈值;并且当多个硬件计算节点中的剩余硬件计算节点小于群体阈值时,重新启动分布式操作系统。

[0354] 实施例16.根据实施例15所述的非暂时性计算机可读存储介质,其上进一步存储有在执行时使一个或多个处理器执行以下操作的指令:设置纪元值以表示分布式操作系统的当前版本;并且将纪元值加1,使得分布式操作系统的实例避免执行两个不同版本的分布式操作系统。

[0355] 实施例17.根据实施例15所述的非暂时性计算机可读存储介质,其上进一步存储有在执行时使一个或多个处理器执行以下操作的指令:标识(identify)被配置为会聚在分布式操作系统的单个版本上的多个硬件计算节点的群体;将多个硬件计算节点的群体中的一个硬件计算节点选择为纪元管理器主控硬件计算节点;以及由该纪元管理器主控硬件计算节点确定多个硬件计算节点的群体内的连接是否已出现故障。

[0356] 实施例18.根据实施例17所述的非暂时性计算机可读存储介质,其中,群体阈值等于多个硬件计算节点的群体中的数量除以2加1((N/2)+1),其中,N表示多个硬件计算节点的群体中的数量。

[0357] 实施例19.根据实施例17所述的非暂时性计算机可读存储介质,其上进一步存储有在执行时使一个或多个处理器执行以下操作的指令:由参与群体的多个硬件计算节点中的一个确定参与群体的多个硬件计算节点中的一个已丧失到该群体的连接,并且响应于多个硬件计算节点中的一个已丧失到群体的连接的确定,由参与群体的多个硬件计算节点中的一个重新启动参与群体的多个硬件计算节点中的一个。

[0358] 实施例20.根据实施例17所述的非暂时性计算机可读存储介质,其上进一步存储有在执行时使一个或多个处理器执行以下操作的指令:在连接故障不超过连接故障阈值时执行当前版本的分布式操作系统。

[0359] 而且,上述任何实施例中阐述的任何特定特征可以组合成所描述技术的有益实施例。即,任何具体特征通常适用于本发明的所有实施例。已经描述了本发明的各种实施例。

[0360] 这些和其他实施方式在所附权利要求的范围内。

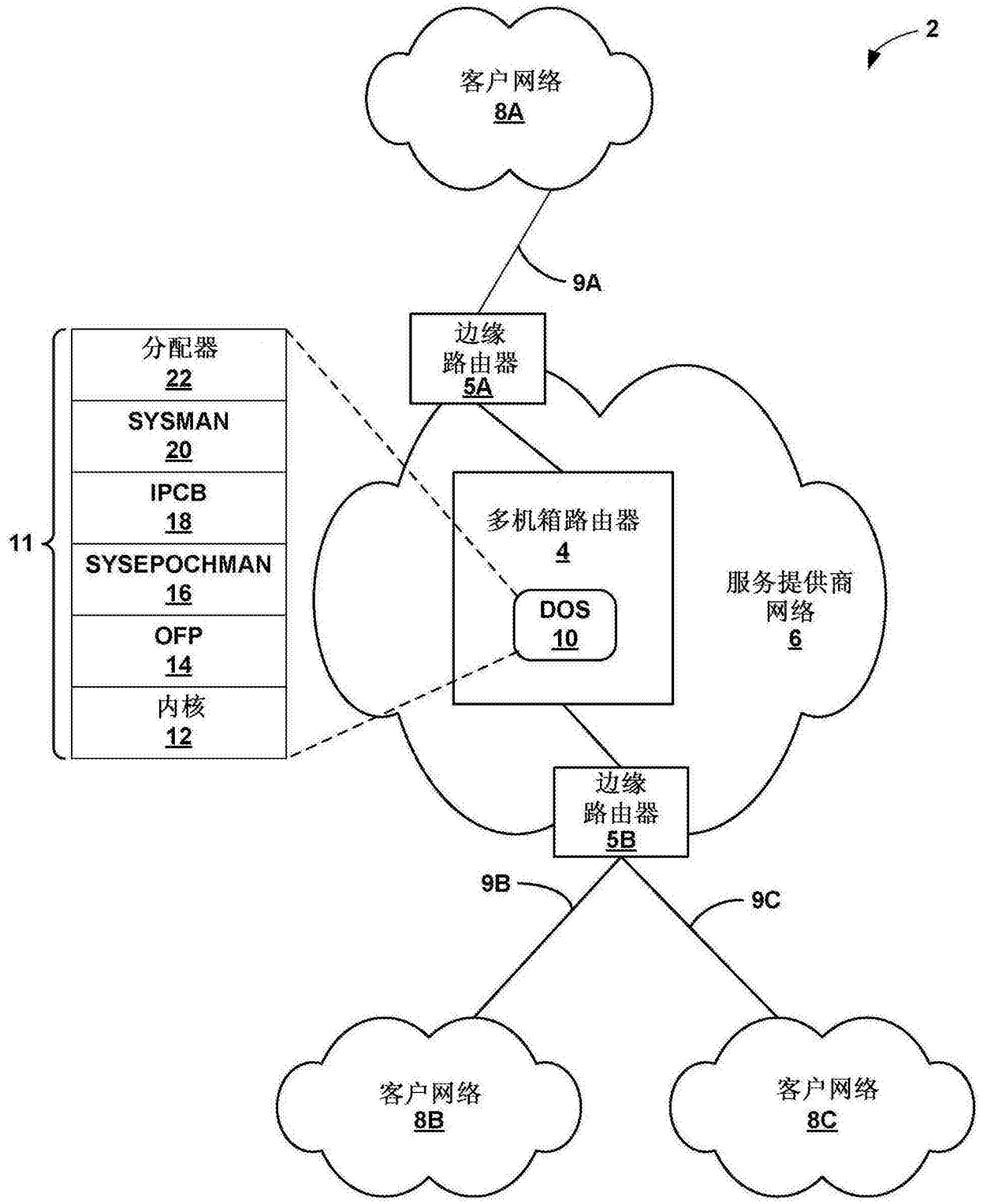

2
客户网络
8A
9A
边缘
路由器
5A
分配器
22
SYSMAN
20
IPCB
18
SYSEPOCHMAN
16
OFP
14
内核
12
11
多机箱路由器
4
DOS
10
服务提供商
网络
6
边缘
路由器
5B
9B
9C
客户网络
8B
客户网络
8C

图1

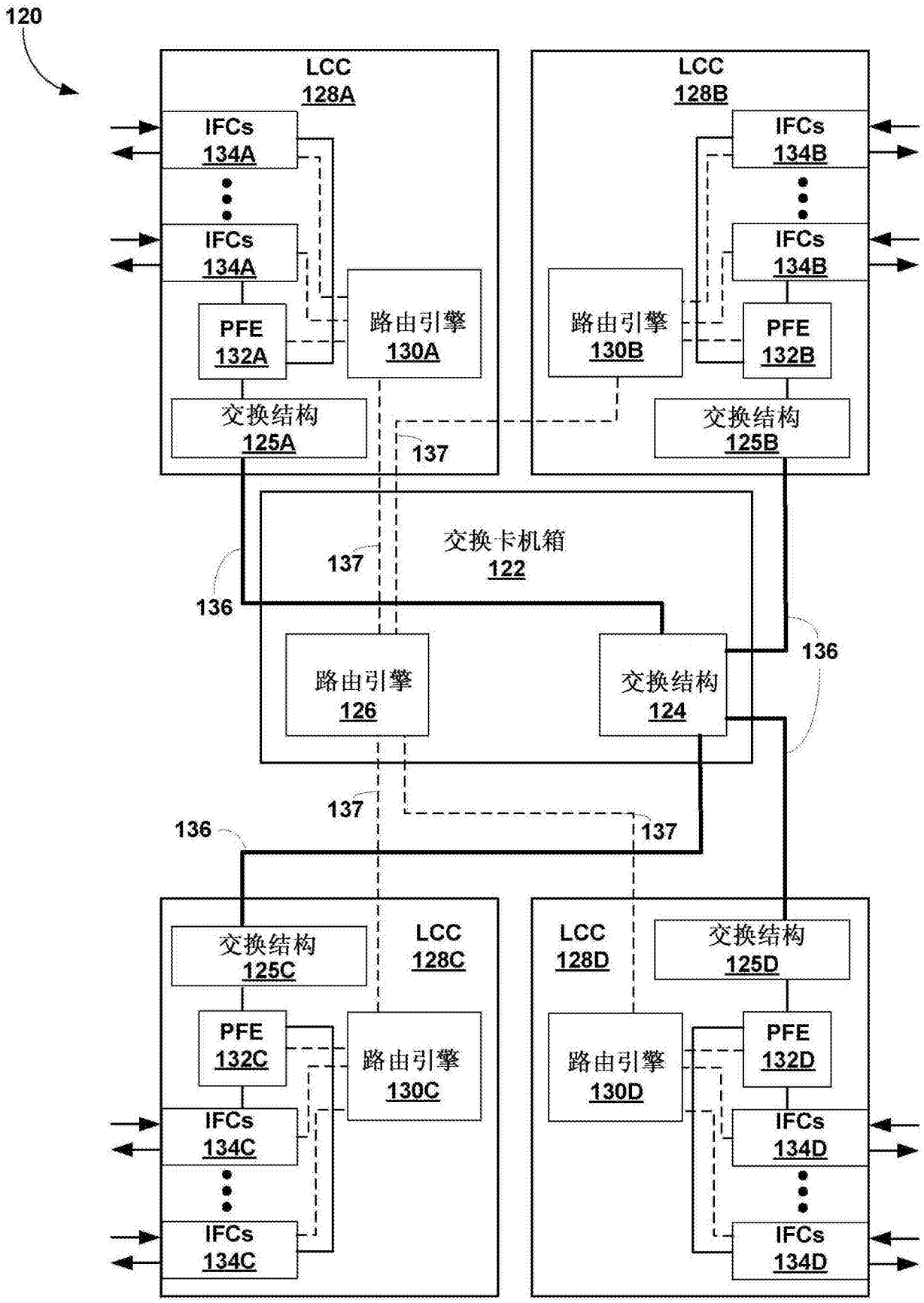

120
LCC
128A
IFCs
134A
IFCs
134A
PFE
132A
路由引擎
130A
交换结构
125A
137
LCC
128B
IFCs
134B
IFCs
134B
路由引擎
130B
PFE
132B
交换结构
125B
交换卡机箱
122
137
136
路由引擎
126
交换结构
124
136
137
137
136
交换结构
125C
LCC
128C
PFE
132C
路由引擎
130C
IFCs
134C
IFCs
134C
LCC
128D
交换结构
125D
PFE
132D
路由引擎
130D
IFCs
134D
IFCs
134D

图2

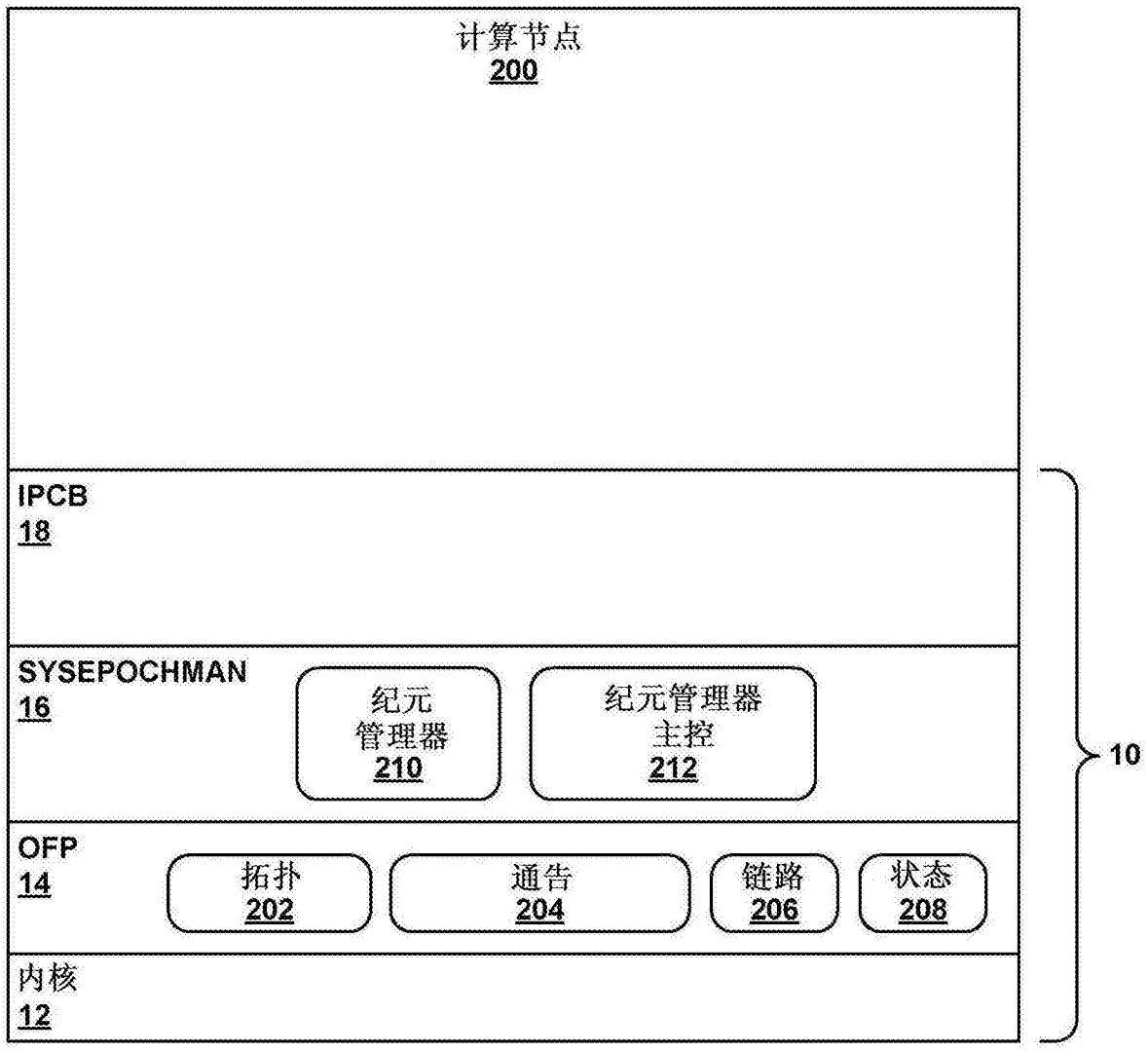

计算节点
200
IPCB
18
SYSEPOCHMAN
16
纪元
管理器
210
纪元管理器
主控
212
OFP
14
拓扑
202
通告
204
链路
206
状态
208
内核
12
10

图3

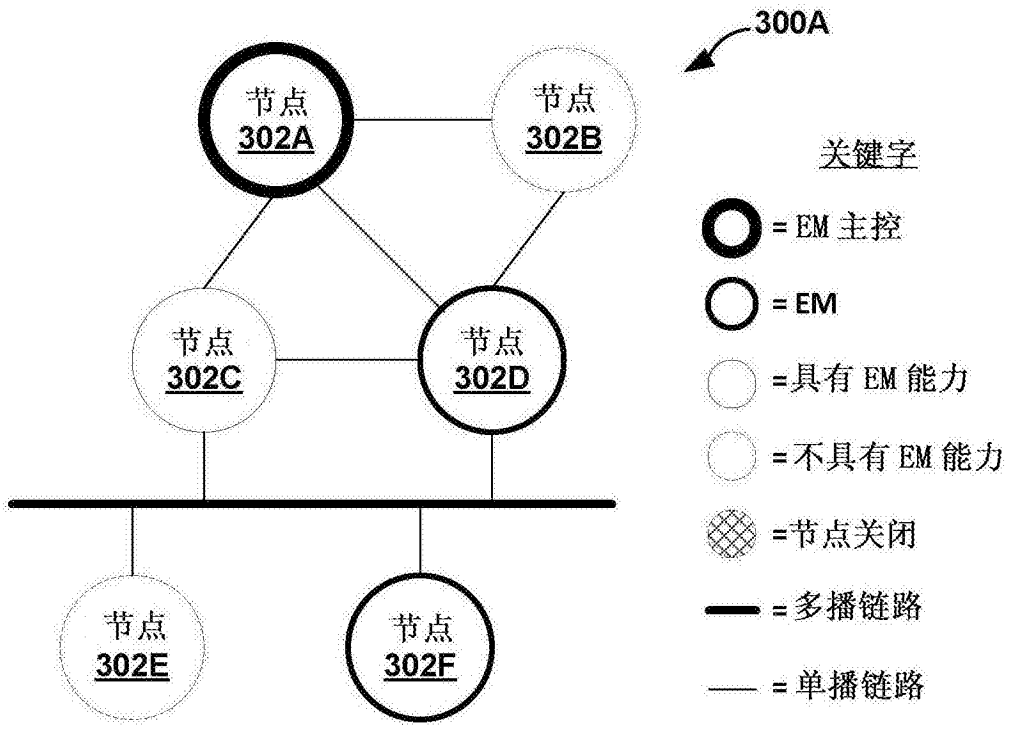

300A
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F
关键字
= EM 主控
= EM
=具有 EM 能力
=不具有 EM 能力
=节点关闭
= 多播链路
= 单播链路

图4A

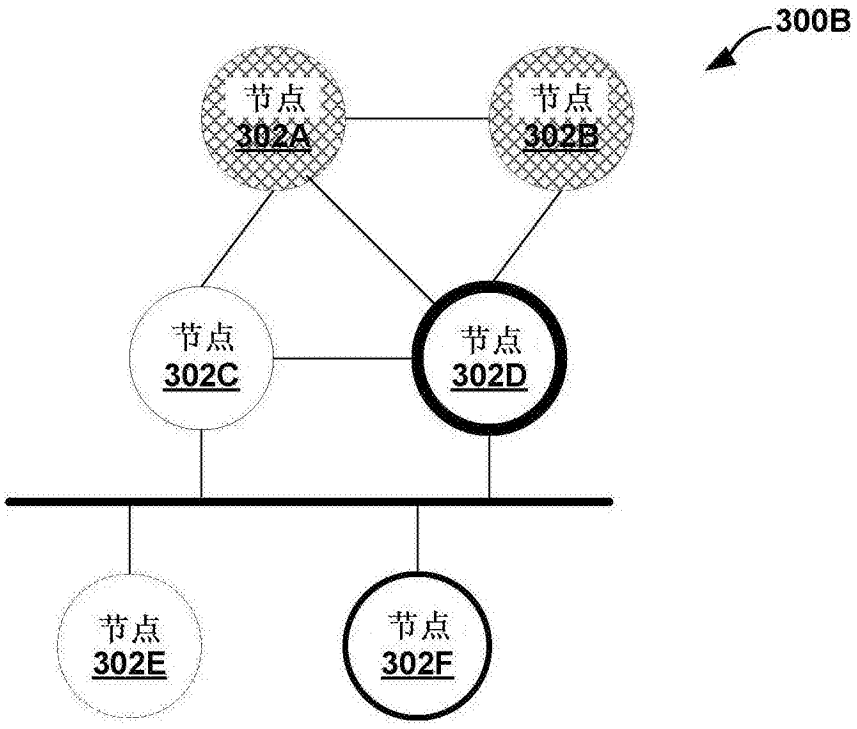

300B
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F

图4B

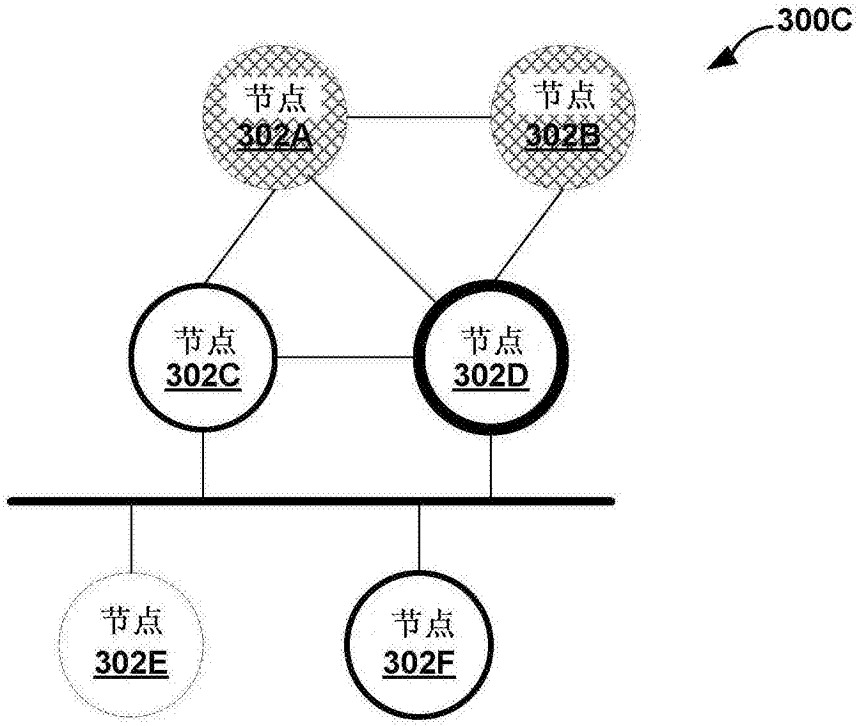

300C
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F

图4C

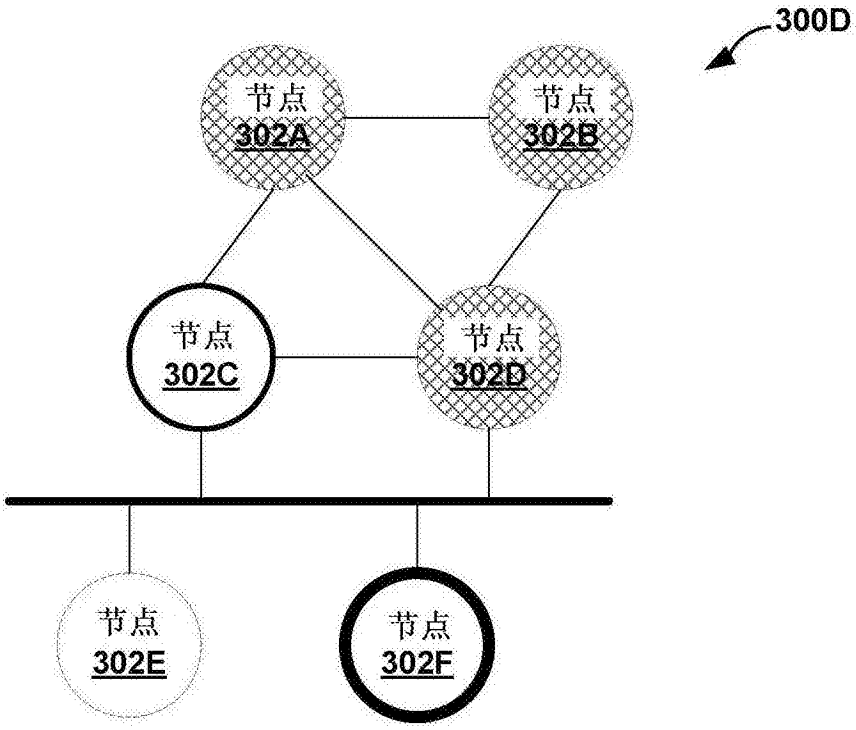

300D
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F

图4D

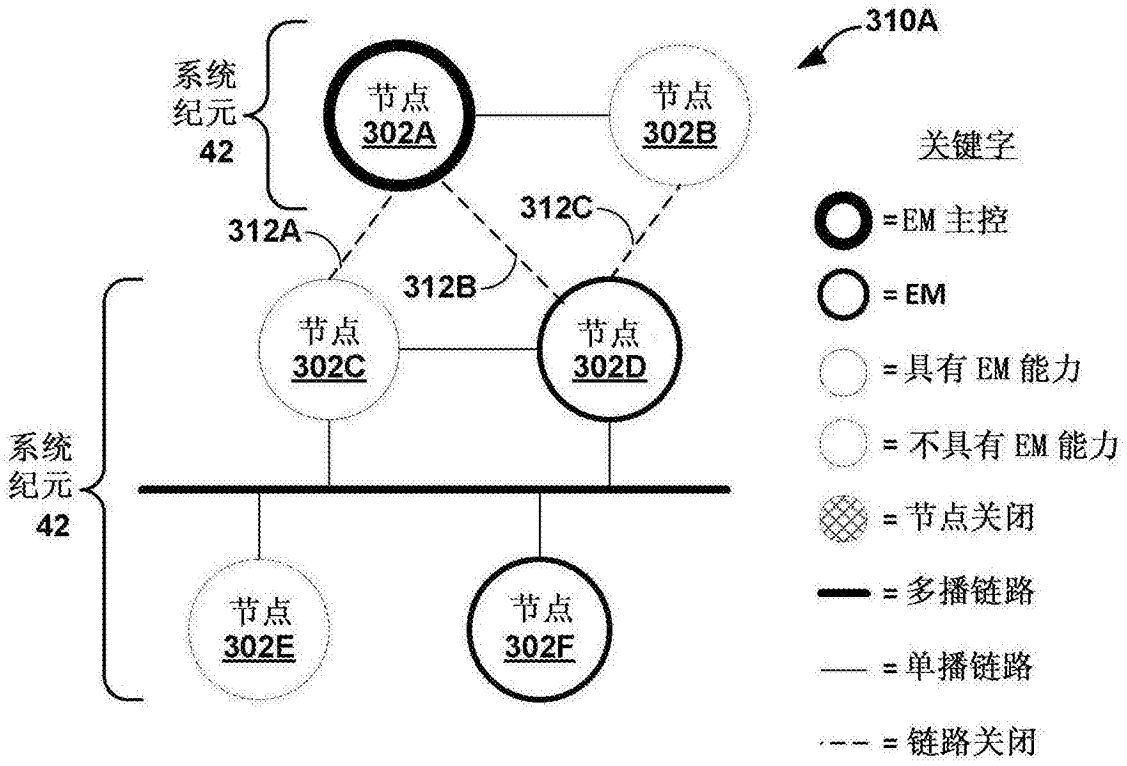

310A
系统
纪元
42
节点
302A
节点
302B
312A
312B
312C
节点
302C
节点
302D
系统
纪元
42
节点
302E
节点
302F
关键字
=EM 主控
=EM
=具有 EM 能力
=不具有 EM 能力
=节点关闭
=多播链路
=单播链路
=链路关闭

图5A

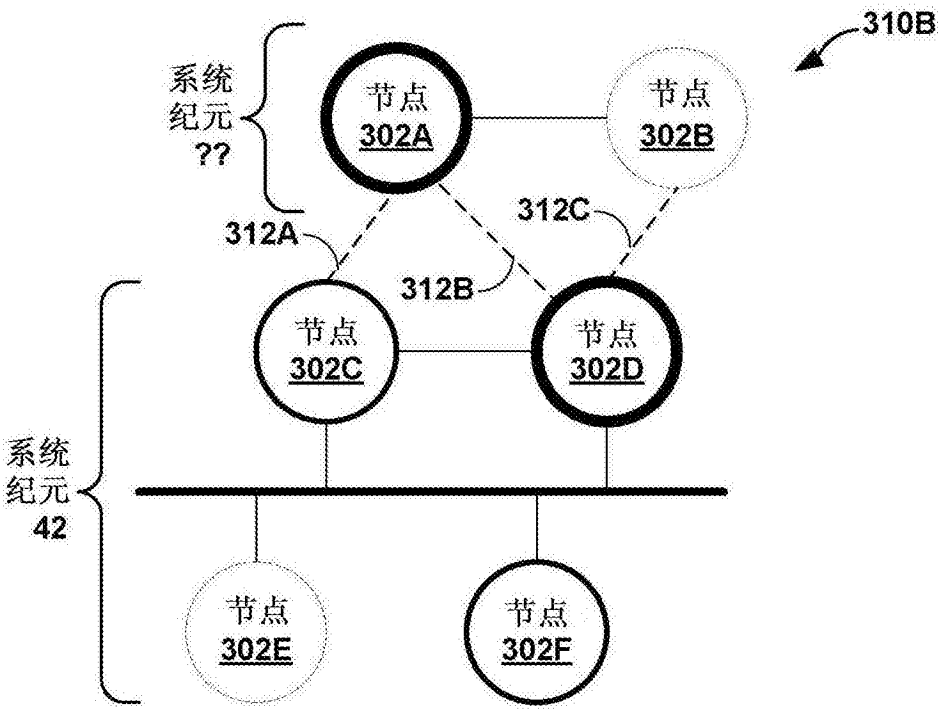

310B
系统
纪元
??
节点
302A
节点
302B
312A
312B
312C
节点
302C
节点
302D
系统
纪元
42
节点
302E
节点
302F

图5B

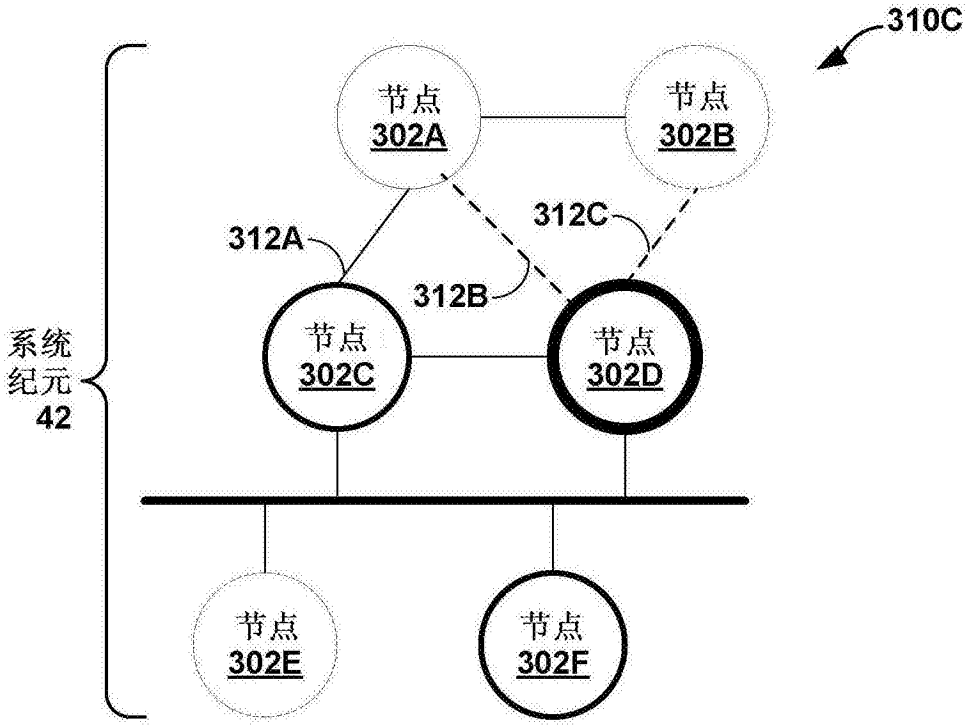

310C
节点
302A
节点
302B
312A
312C
312B
节点
302C
节点
302D
系统
纪元
42
节点
302E
节点
302F

图5C

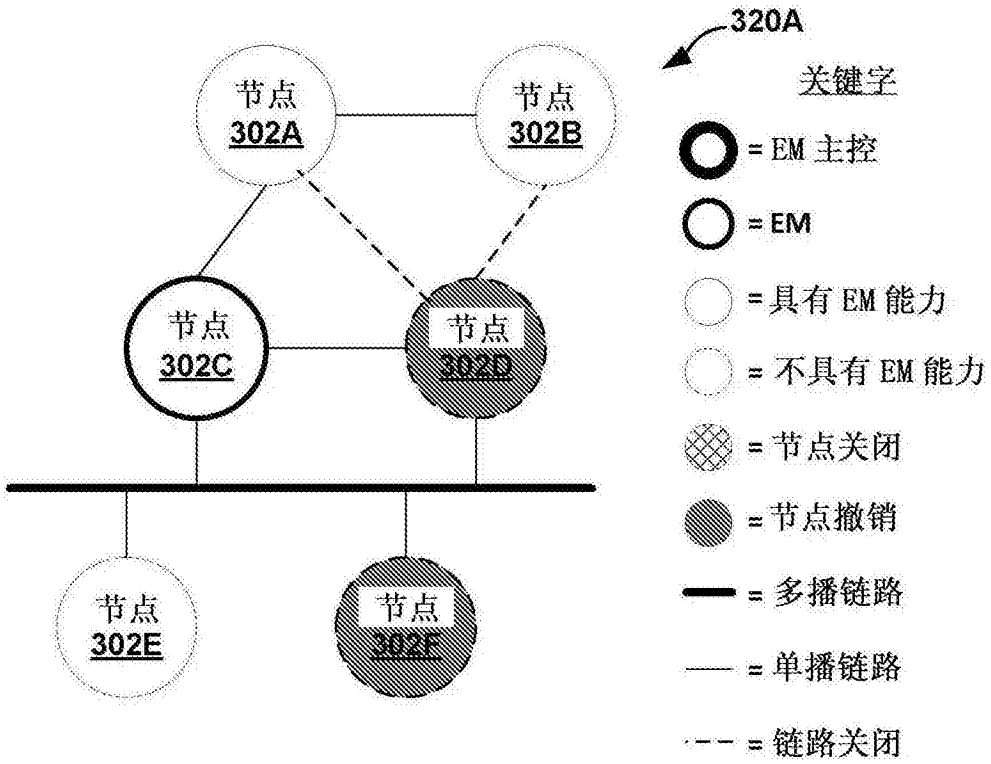

320A
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F
关键字
= EM 主控
= EM
= 具有 EM 能力
= 不具有 EM 能力
= 节点关闭
= 节点撤销
= 多播链路
= 单播链路
= 链路关闭

图6A

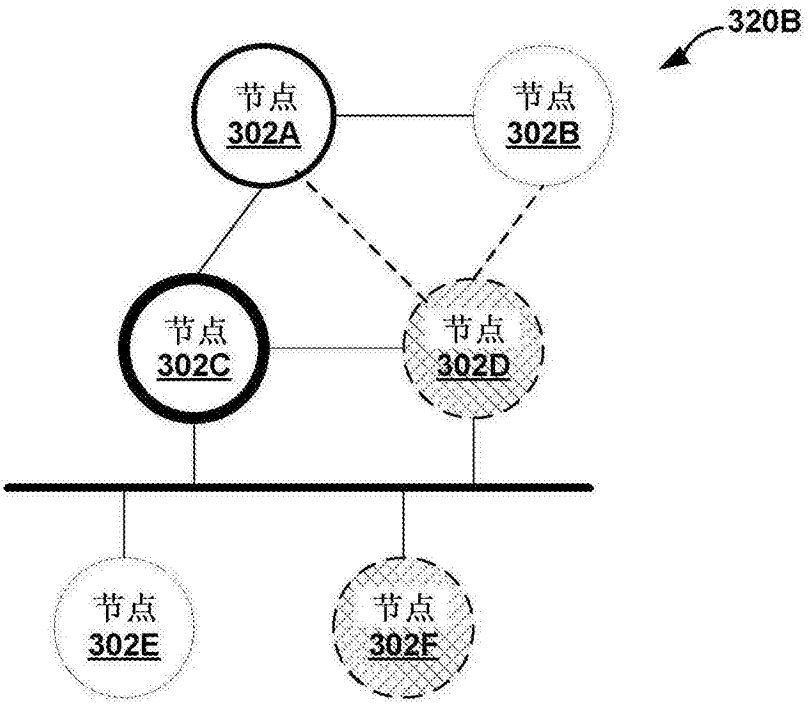

320B
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F

图6B

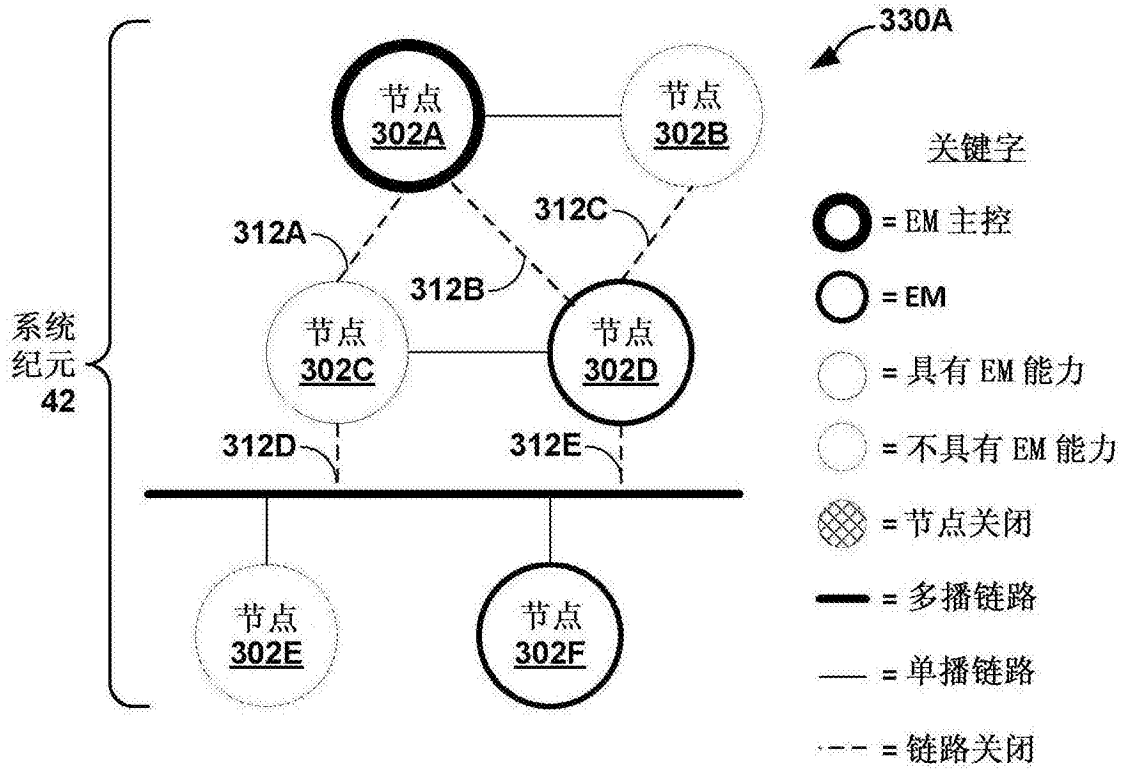

330A
系统
纪元
42
节点
302A
节点
302B
312A
312B
312C
节点
302C
节点
302D
312D
312E
节点
302E
节点
302F
关键字
= EM 主控
= EM
= 具有 EM 能力
= 不具有 EM 能力
= 节点关闭
= 多播链路
= 单播链路
= 链路关闭

图7A

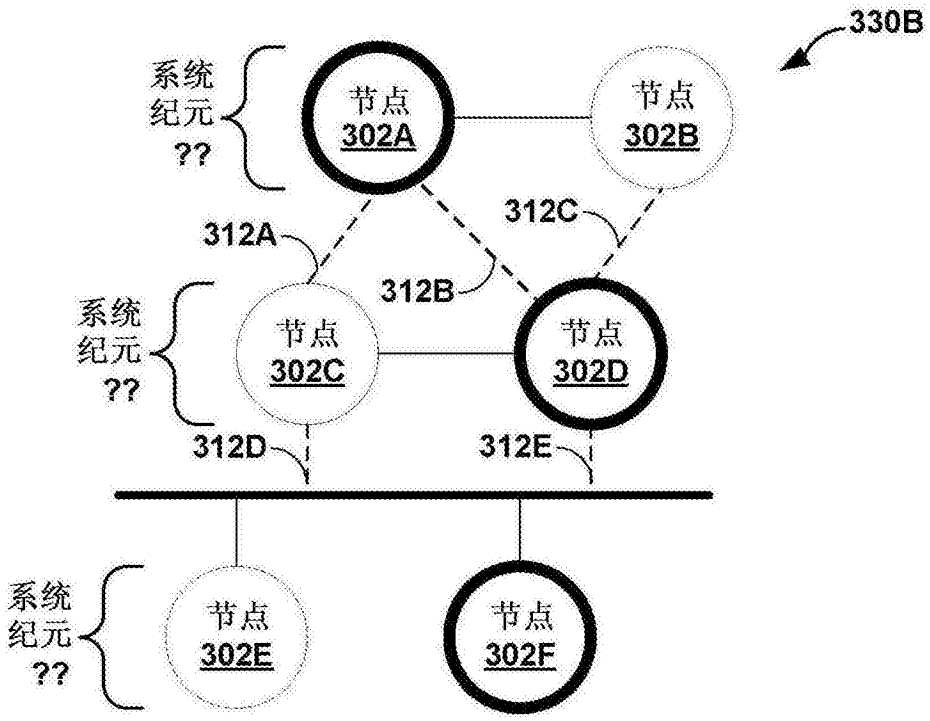

330B
系统
纪元
??
节点
302A
节点
302B
312A
312B
312C
系统
纪元
??
节点
302C
节点
302D
312D
312E
系统
纪元
??
节点
302E
节点
302F

图7B

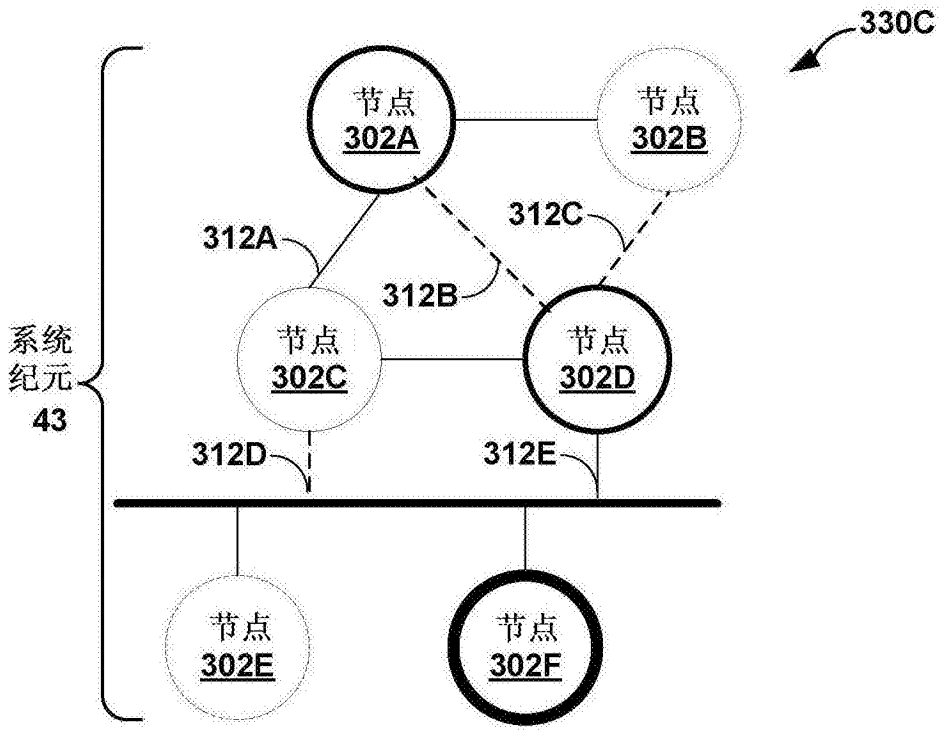

330C
节点
302A
节点
302B
312A
312B
312C
系统
纪元
43
节点
302C
节点
302D
312D
312E
节点
302E
节点
302F

图7C

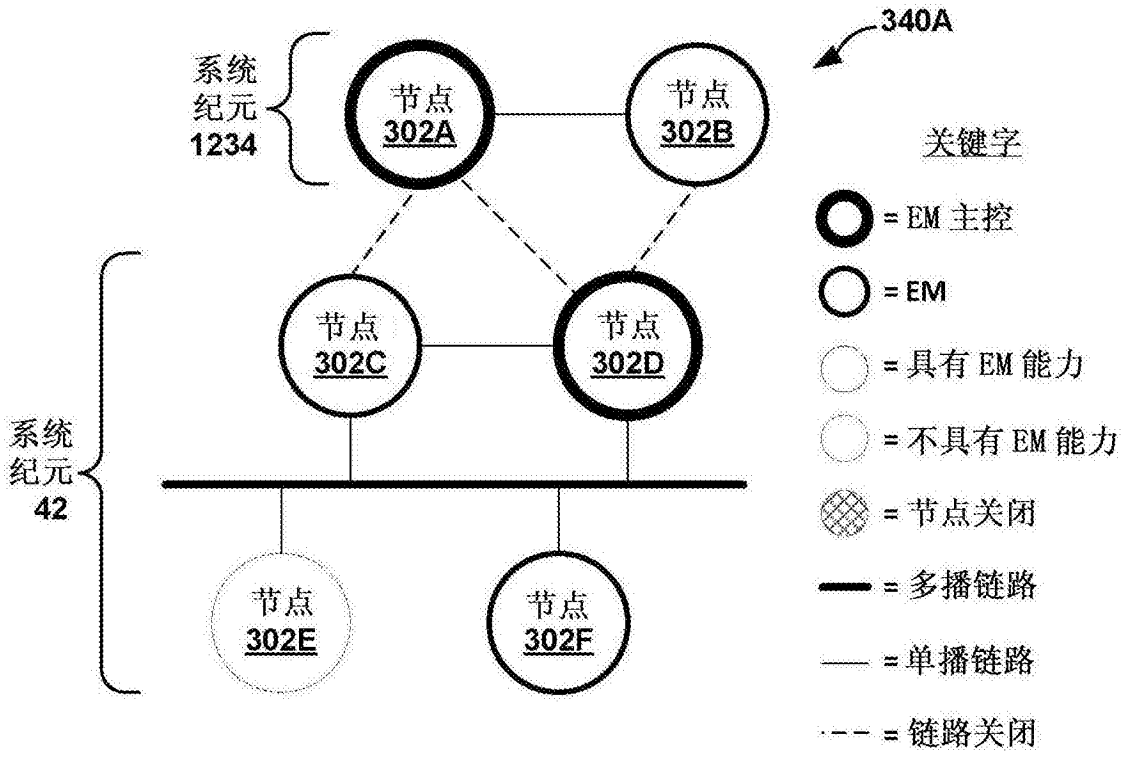

340A
系统
纪元
1234
节点
302A
节点
302B
节点
302C
节点
302D
系统
纪元
42
节点
302E
节点
302F
关键字
= EM 主控
= EM
= 具有 EM 能力
= 不具有 EM 能力
= 节点关闭
= 多播链路
= 单播链路
= 链路关闭

图8A

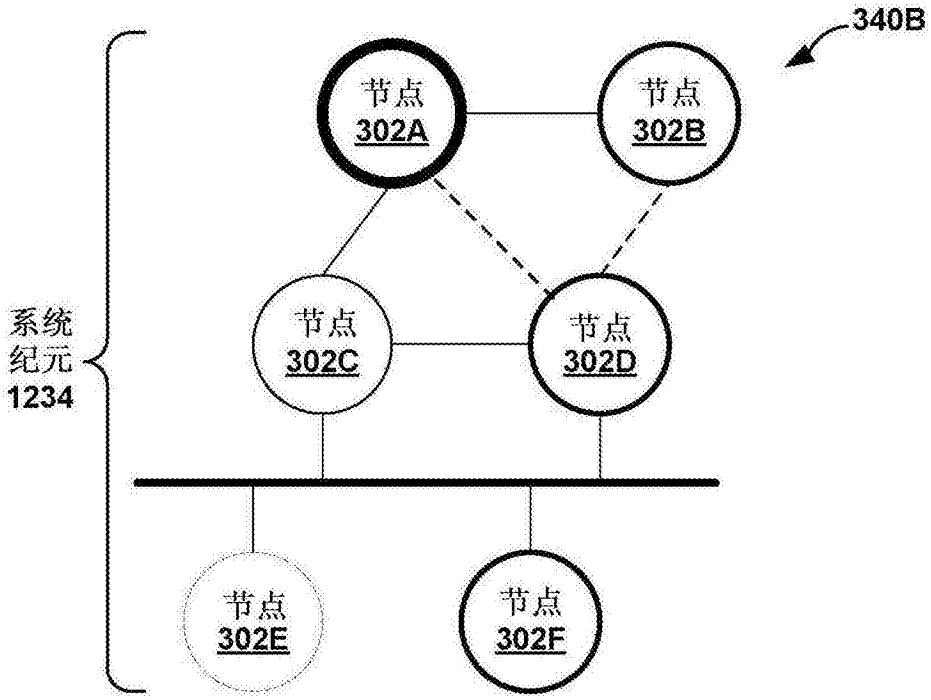

340B
系统
纪元
1234
节点
302A
节点
302B
节点
302C
节点
302D
节点
302E
节点
302F

图8B

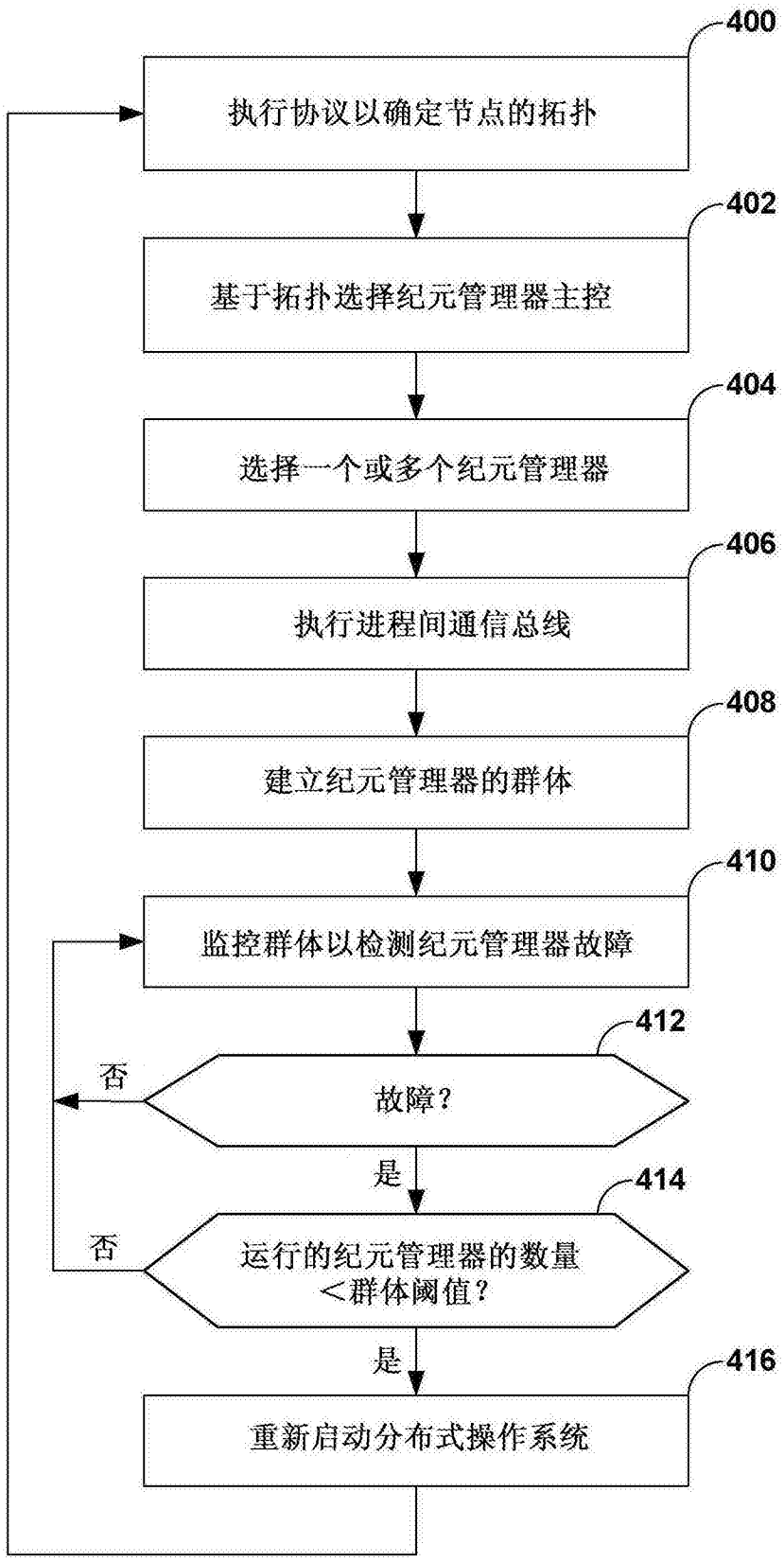

400
执行协议以确定节点的拓扑
402
基于拓扑选择纪元管理器主控
404
选择一个或多个纪元管理器
406
执行进程间通信总线
408
建立纪元管理器的群体
410
监控群体以检测纪元管理器故障
412
故障？
否
是
414
运行的纪元管理器的数量
＜群体阈值？
否
是
416
重新启动分布式操作系统

图9

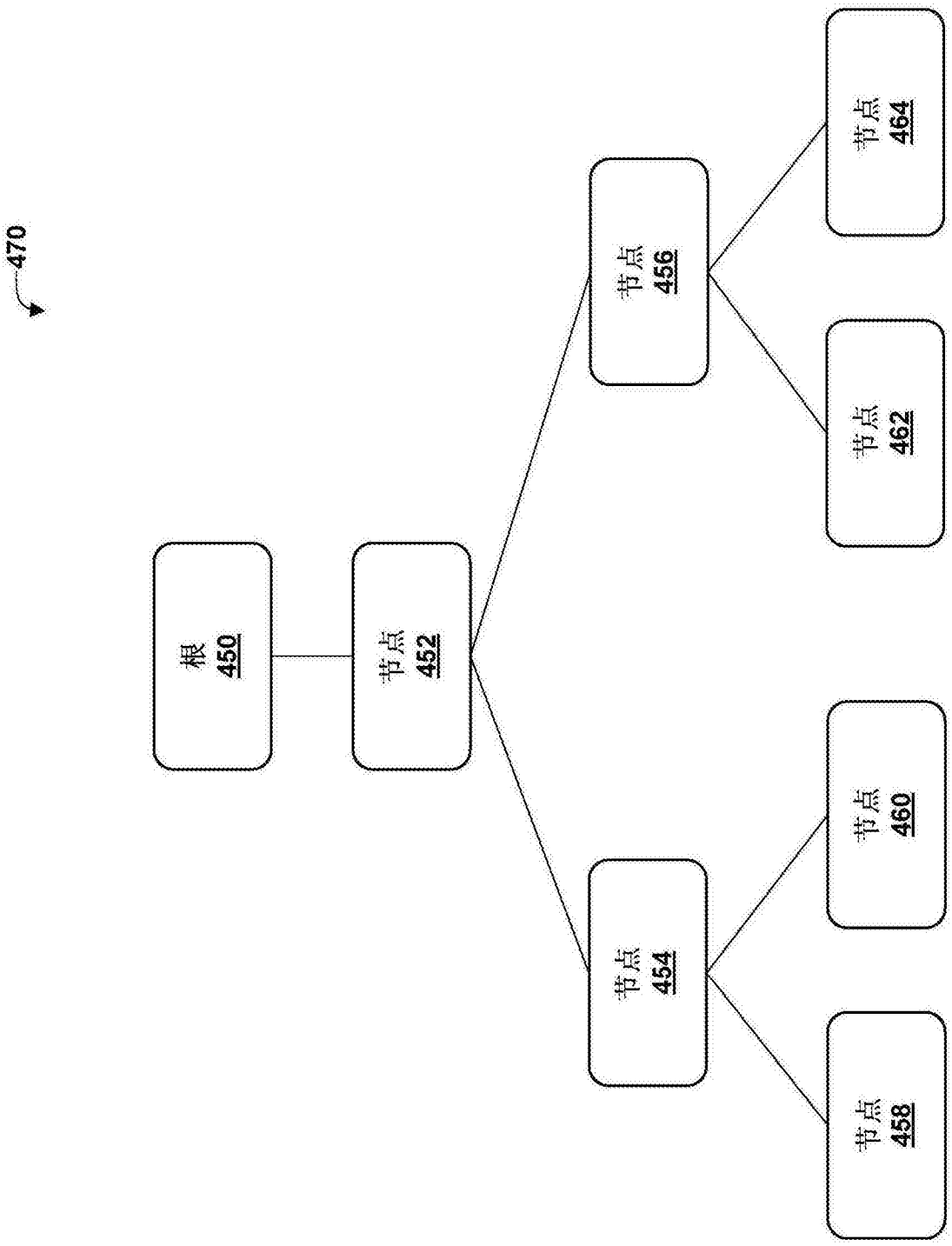


图10

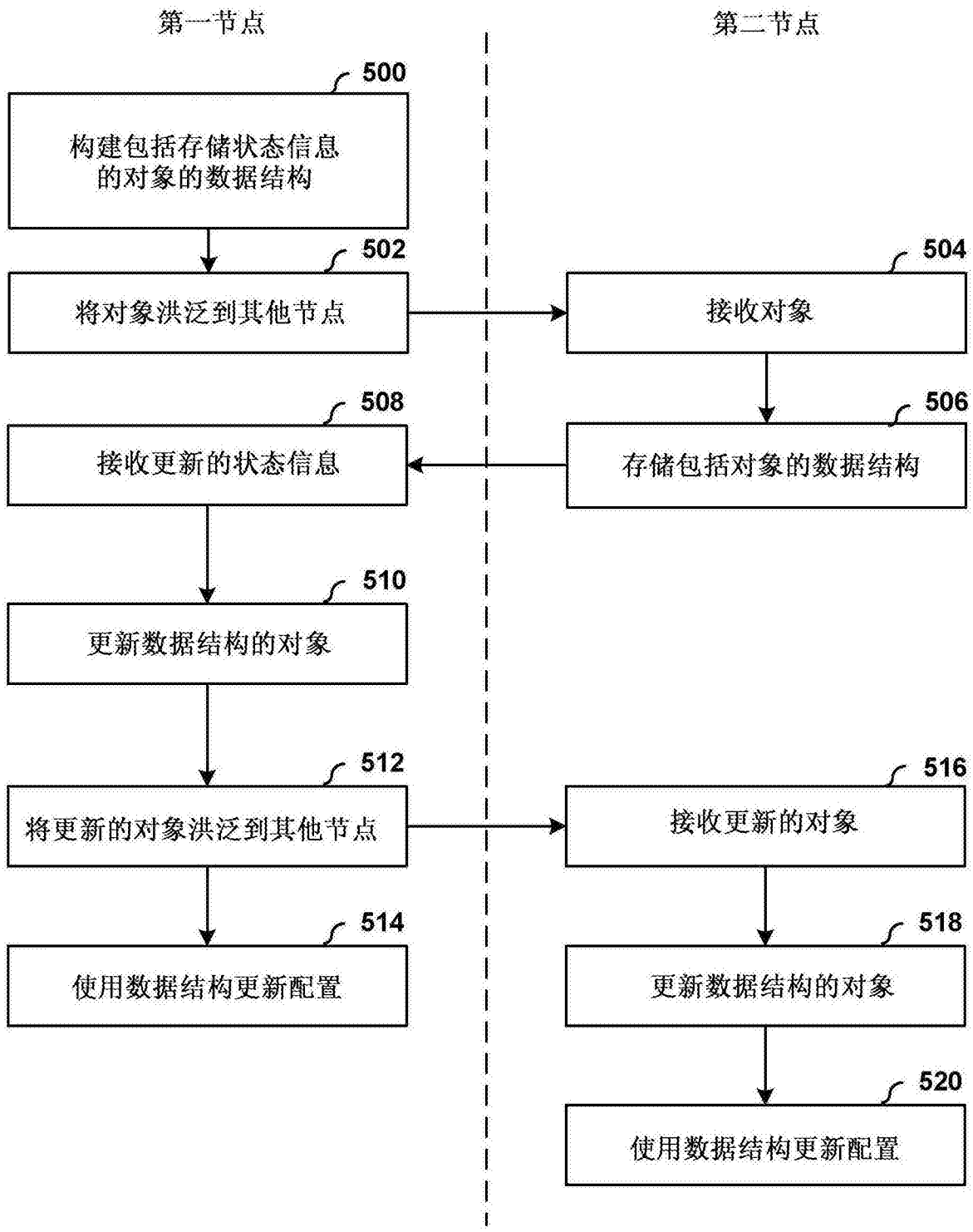

第一节点
第二节点
500
构建包括存储状态信息的对象的数据结构
502
将对象洪泛到其他节点
504
接收对象
506
存储包括对象的数据结构
508
接收更新的状态信息
510
更新数据结构的对象
512
将更新的对象洪泛到其他节点
514
使用数据结构更新配置
516
接收更新的对象
518
更新数据结构的对象
520
使用数据结构更新配置

图11