(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 1955931 B
(45)授权公告日 2010.10.06

(21)申请号 200610131758.4
(22)申请日 2006.09.30
(30)优先权数据
0519981.5 2005.09.30 GB
(73)专利权人 科威尔公司
地址 美国加利福尼亚州
专利权人 富士通微电子株式会社
(72)发明人 马克·大卫·李佩特
(74)专利代理机构 北京三友知识产权代理有限公司 11127
代理人 孙海龙
(51)Int.Cl.
G06F 9/46(2006.01)
(56)对比文件
US 7159216 B2, 全文.
CN 1402471 A, 2003.03.12, 全文.

审查员 马雅凡

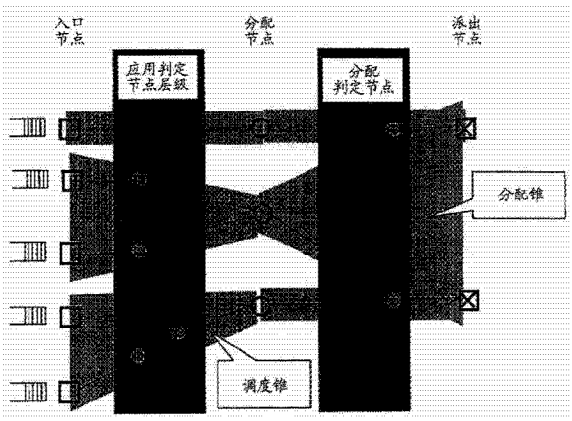
权利要求书 4 页 说明书 44 页 附图 46 页

(54)发明名称

在多核处理器内调度可执行事务的方法及多核处理器

(57)摘要

本发明涉及在多核架构中的调度。本发明涉及在多核处理器中调度线程。可以使用至少一个分配队列和多级调度器来调度可执行事务，所述分配队列按对执行的适任性的顺序排列可执行事务，所述多级调度器包括多个链接的个体可执行事务调度器。它们每个包括一调度算法，用于确定用于执行的最适任可执行事务。从多级调度器将最适合可执行事务输出到所述至少一个分配队列。



CN 1955931 B

1. 一种在多核处理器内调度可执行事务的方法，所述多核处理器具有多个处理器元件，所述方法包括：

设置一个分配队列，所述分配队列按可执行事务执行的适任性顺序排列可执行事务；

设置多个个体可执行事务调度器，其中，每个个体可执行事务调度器包括调度算法，所述调度算法用于从准备好执行的多个候选可执行事务中确定用于执行的最适任可执行事务；

将所述多个调度器链接在一起，从而提供多级调度器；

从所述多级调度器将所述最适任可执行事务输出到所述分配队列；

设置多个派出队列，其包括用于每个处理器元件的派出队列，每个派出队列按在相应的处理器元件上的执行顺序排列所述可执行事务；

将所述分配队列与所述多个派出队列的一个或更多个派出队列相关联；以及

将所述分配队列的内容输出到一个或更多个相关联的派出队列。

2. 根据权利要求 1 所述的方法，其中，所述多级调度器具有分层结构。

3. 根据权利要求 1 或 2 所述的方法，其中，所述可执行事务中的一些或全部一起定义了一应用，所述方法还包括：

在所述应用的运行时间期间开始执行可执行事务；

在所述运行时间期间，动态地创建新的个体可执行事务调度器；以及

将所述新的个体可执行事务调度器链接到所述多级分层调度器。

4. 根据权利要求 3 所述的方法，其中，用于每个个体可执行事务调度器的调度算法包括至少一种规则，以按照用于在所述多个处理器元件的一个或更多个处理器元件上执行的最适任事务的顺序来排列所述可执行事务。

5. 根据权利要求 4 所述的方法，其中，链接新的动态创建的个体可执行事务调度器的步骤包括：

将准备好执行的一个或更多个可执行事务分配给所述新的个体可执行事务调度器。

6. 根据权利要求 4 所述的方法，该方法还包括：

在所述动态创建的个体可执行事务调度器完成了对所标记的所有可执行事务的处理时，对所述动态创建的个体可执行事务调度器解除链接；以及

从所述多级分层调度器中移除所述个体可执行事务调度器。

7. 根据权利要求 1 所述的方法，该方法还包括：

设置至少一个待定队列，用于将所述候选可执行事务提供给所述多个个体可执行事务调度器。

8. 根据权利要求 1 所述的方法，其中将所述分配队列与一特定处理器元件相关联，并且其中输出步骤包括将所述分配队列的内容直接输出到相关联的处理器元件的派出队列中。

9. 根据权利要求 1 所述的方法，其中关联步骤包括将所述分配队列与多个处理器元件相关联，并且其中输出步骤包括根据与所述派出队列相关联的处理器元件的适任性，将所述分配队列的内容输出到相关联的处理器元件的一个或更多个派出队列，以执行特定可执行事务，从而设置了处理器元件的池。

10. 根据权利要求 9 所述的方法，该方法还包括：

设置第二个分配队列；以及

将所述第二个分配队列的内容输出到所述处理器元件的池的一个派出队列。

11. 根据权利要求 9 或 10 所述的方法，该方法还包括：

设置另一个分配队列；以及

通过将所述另一个分配队列与另外的多个处理器元件相关联，并且根据与所述派出队列相关联的处理器元件的适任性，将所述另一个分配队列的内容输出到相关联的处理器元件的一个或更多个派出队列，以执行特定可执行事务，从而设置了处理器元件的另一池。

12. 根据权利要求 9 所述的方法，其中，执行特定可执行事务的相关联处理器元件的适任性取决于所述处理器元件的繁忙状态。

13. 根据权利要求 12 所述的方法，其中，处理器元件的所述繁忙状态取决于已经位于所述执行特定可执行事务的相关联处理器元件的派出队列中的可执行事务的数量。

14. 根据权利要求 9 所述的方法，其中，执行特定可执行事务的相关联处理器元件的适任性取决于所述处理器元件的功率状态。

15. 根据权利要求 11 所述的方法，其中，一个或更多个所述处理器元件位于处理器元件的多个池中。

16. 根据权利要求 11 所述的方法，该方法还包括这样的步骤：在处理器元件的池之间移动可执行事务。

17. 根据权利要求 1 所述的方法，该方法还包括：

设置可执行事务的至少一个时间片组，所述时间片组与所述处理器元件中的一个相关联，并且所述时间片组中的每个可执行事务在相关联处理器元件上具有预定义的执行时间部分。

18. 根据权利要求 9 所述的方法，该方法还包括：

设置可执行事务的至少一个时间片组，所述时间片组与处理器元件的池相关联，并且所述时间片组中的每个可执行事务在处理器元件的相关联池上具有预定义的执行时间部分。

19. 根据权利要求 17 或 18 所述的方法，该方法还包括：

针对可执行事务的每个时间片组，设置可执行事务的一抢占组，可执行事务的所述抢占组相对于可执行事务的所述时间片组中的任何一个，在相关联的处理器元件或处理器元件的池上具有执行优先级。

20. 根据权利要求 19 所述的方法，该方法还包括：

当可执行事务的所述抢占组在处理器元件的相关联池中的其他处理器元件中的至少一个上执行时，提供在处理器元件的相关联池中的处理器元件中的至少一个上具有预定义的执行时间部分的可执行事务的所述至少一个时间片组。

21. 根据权利要求 20 所述的方法，该方法还包括：

使来自所述至少一个时间片组的可执行事务在相关联处理器元件上执行；

使所述可执行事务对来自所述抢占组的可执行事务退让；以及

当来自所述抢占组的可执行事务的执行完成时，在相关联处理器元件上继续执行来自所述至少一个时间片组的所述可执行事务。

22. 根据权利要求 9 所述的方法，其中，处理器元件的池内的每个处理器元件具有至少

一个节能模式，并且所述方法还包括：

当分配给处理器元件的池的可执行事务的数量使得可用处理器元件的仅仅一部分用于执行可执行事务时，将所述处理器元件的池内的一个或更多个所述处理器元件置入第一节能模式。

23. 根据权利要求 22 所述的方法，其中，所述处理器元件的池内的每个处理器元件具有至少两个节能模式，并且所述方法还包括：

在预定义的时间段后，将所述处理器元件的池内的一个或更多个处理器元件置入第二节能模式，其中所述第二节能模式比所述第一节能模式节约更多能量。

24. 根据权利要求 22 或 23 所述的方法，其中，按预定义的顺序分配所述处理器元件的池内的处理器元件，并且用于将可执行事务分配给所述处理器元件的池内的处理器元件的调度算法总是将可执行事务分配给在所述预定义的顺序中第一个可用的处理器元件。

25. 根据权利要求 22 所述的方法，该方法还包括：

响应于分配给处理器元件的池的可执行事务的数量的增加，使所述处理器元件的池内的当前处于节能模式下的一个或更多个处理器元件返回到正常电力模式。

26. 根据权利要求 22 所述的方法，其中，通过增加可执行事务的最小执行优先级来发起所述至少一个节能模式，该可执行事务将由处理器元件执行。

27. 根据权利要求 1 所述的方法，该方法还包括：

识别可执行事务以及多核处理器中至少一方的变化；以及

作为识别到变化的结果，针对个体可执行事务调度器，从多个准备好执行的候选可执行事务中重新确定用于执行的最适任可执行事务。

28. 根据权利要求 27 所述的方法，其中，针对受所识别出的变化所影响的每个个体可执行事务调度器执行重新确定的步骤。

29. 根据权利要求 1 所述的方法，其中，所述多核处理器内的至少一个处理器元件包含可重新配置的逻辑，所述可重新配置的逻辑在运行时间期间可重新配置，所述方法还包括：

为可重新配置的逻辑的每个配置提供可执行事务的配置队列，所述配置队列列出准备好执行并且分配给所述可重新配置的处理器元件的特定配置的可执行事务；

将与当前选择的配置实例相关联的配置队列的内容输出到所述可重新配置的处理器元件以用于执行；以及

当到达为了保持应用的完整性而必须执行特定可执行事务之前所允许的最大时间时，切换当前选择的配置实例。

30. 根据权利要求 29 所述的方法，其中，切换配置实例的步骤还包括：

停止对与当前选择的配置相关联的配置队列的内容的输出；

选择可用配置实例中的另一个，并且将可重新配置的逻辑的配置改变为可用配置实例中的所述选择的另一个；以及

将与新选择的配置实例相关联的配置队列的内容输出到所述可重新配置的处理器元件以用于执行。

31. 根据权利要求 29 或 30 所述的方法，其中，所述可重新配置的逻辑是存储器。

32. 根据权利要求 30 所述的方法，其中，对另一配置实例的选择依赖于各个配置队列

的长度。

33. 根据权利要求 32 所述的方法，其中，选择与最长的配置队列相关联的配置实例。

34. 根据权利要求 1 所述的方法，其中，所述多个个体可执行事务调度器使用至少两个不同的调度算法。

35. 根据权利要求 34 所述的方法，其中，所使用的调度算法包括：

先入先出调度；基于优先级的调度；轮转调度；或加权公平队列调度。

36. 根据权利要求 1 所述的方法，其中，使用专用存储器中的指针来进行所述多核处理器内的可执行事务的操作和 / 或分配，所述指针引用主存储器，所述主存储器包含指令，所述指令包括所述可执行事务。

37. 根据权利要求 36 所述的方法，其中，所述分配队列或派出队列是存储在所述专用存储器中的指针的链接集。

38. 根据权利要求 36 或 37 所述的方法，其中，所述指针包括多个可执行事务的参数，所述可执行事务的参数定义所述可执行事务的特征，根据所述特征做出调度判定，并且其中，所述可执行事务的参数用于调度可执行事务。

39. 根据权利要求 38 所述的方法，其中，所述调度算法随时间操纵所述参数。

40. 根据权利要求 38 所述的方法，其中，所述多个可执行事务的参数包括可执行事务的参数的第一和第二子集，并且其中，仅当可执行事务的参数的第一子集位于预定义的配置中时，才利用可执行事务的参数的所述第二子集。

41. 一种多核处理器，所述多核处理器包括：

多个处理器元件；

一个分配队列，所述分配队列按执行的适任性顺序排列可执行事务；

多级调度器，所述多级调度器包括多个个体可执行事务调度器，其中每个个体可执行事务调度器包括调度算法，所述调度算法用于从准备好执行的多个候选可执行事务中确定用于执行的最适任可执行事务；其中将所述多个调度器链接在一起，并将所述多级调度器设置成从所述多级调度器将所述最适任可执行事务输出到所述分配队列；以及

多个派出队列，其包括用于每个处理器元件的派出队列，将每个派出队列设置成按在相应的处理器元件上的执行顺序排列所述可执行事务；以及

其中将所述分配队列与所述多个派出队列的一个或更多个派出队列相关联，并将所述分配队列设置成将所述分配队列的内容输出到一个或更多个相关联的派出队列。

在多核处理器内调度可执行事务的方法及多核处理器

技术领域

[0001] 本发明涉及一种用于在多核架构中调度线程的方法和装置。

背景技术

[0002] 近年来，出现了生产含有多个核的处理器的趋势，以使硅效率（即，“应用可用的”MIP/平方毫米或MIP/毫瓦）最大化。这种多核架构理想地适于运行基于线程的应用，这是因为线程定义了包含执行状态、指令流和数据集的工作的自治封装，通过该定义，其可以与其他线程并发地执行。

[0003] 调度是概括术语，用于对于特定处理资源发现并分配最适合的线程（即指令集）来执行，并且应用程序以及在其上执行应用程序的下层硬件平台都需要调度。

[0004] 与适于执行特定线程的多个核可能的可用性相结合，在多核结构内的执行并发性将另外的问题引入了用于在这些多核架构内分配线程的调度中。

发明内容

[0005] 根据本发明的第一方面，提供了一种在多核处理器内调度可执行事务（更经常地称为线程）的方法，如权利要求1所述。

[0006] 通过提供多级（更优选地为分层）的调度系统，本发明使得能够根据一个或更多个较简单的调度算法构造出复杂的调度算法。使用这种复杂调度算法的能力提高了包括多个线程的应用在其执行期间的性能。本发明通过更有效地将可执行事务或线程分配给处理资源，而提高了应用的运行时性能。这可以提高执行速度并且减少对特定资源的瓶颈。它还可增加对多核处理器的较不活跃部分的使用。

[0007] 在优选实施例中，由专用硬编码（从而有效率）实施例来执行该方法。优选的硬编码实施例是服务器-客户机拓扑结构的，包括系统服务器（下文中称为SystemWeaver）以及多核处理器内每个处理资源或核的客户机。在其他的实施例中，正在讨论的处理资源的能力使单个客户机可以聚合对多个处理资源的访问。

[0008] 为了进一步提高并入了本发明的整个系统的性能，优选实施例使用专用存储器内的指针来调度分配、存储用于判定目的的值等。这些指针优选地包含用于存储根据其做出调度判定的参数的字段，并且优选地还包含这样的字段，该字段用于存储在后续判定中使用的值或者简单地存储关心的其他值。在下文中所述字段集总地被称为量度(metric)或操作符。

[0009] 根据本发明的第二方面，提供了一种在多核处理器内调度可执行事务的方法，所述可执行事务定义了应用，并且所述多核处理器具有多个处理器元件，所述方法包括：保持可执行事务调度器的层级，其中，所述层级适于根据所述应用在使用时的要求来调度可执行事务，其中，所述调度器层级的每个级别包括至少一个调度器，并且其中，所述至少一个调度器包括至少一个规则，用于将所述可执行事务排序成在一个或更多个处理器元件上执行的最适任事务的顺序。

[0010] 根据本发明的第三方面，提供了一种在多核处理器中管理能耗的方法，如权利要求 37 所述。

[0011] 在多核处理器具有多个处理资源、每个处理资源都能够执行特定线程的情况下，本发明使这多个处理资源能够一起置入池中。因而是该处理资源池被分配了线程。然而，在这种情况下，当需要执行的线程的数量未超过池的执行能力时，即，当该池内的某些处理资源要么被利用，要么完全未被利用时，本发明使各个处理资源能够置入节能模式。处理资源甚至可具有多个不同的节能级别。

[0012] 优选地，当线程执行负载需要时，处理资源离开节能模式，或者至少将处理资源移动到较不节能的模式，其返回到全功率模式的代价较低。

[0013] 根据本发明的第四方面，如权利要求 38 所述，提供了一种在多核处理器内调度可执行事务或线程的方法，所述多核处理器具有至少一个包括可重新配置的逻辑的处理器元件。

[0014] 在一个或更多个处理器元件具有可重新配置的逻辑部分的情况下，例如在场可编程门阵列 (FPGA) 的可重新配置的执行部分的情况下，本发明可以通过将利用该逻辑的相同配置的线程聚合在一起来提高性能。即，当可重新配置的逻辑部分被重新配置时，这用来减少环境 (context) 切换的需要或降低环境切换的影响。

[0015] 本发明还可在可重新配置的逻辑的情况下提高性能，形式为，局部高速缓冲存储器存储准备好接下来在所讨论的处理器元件上执行的指令。

[0016] 这是因为通过将使用高速缓冲的存储器中相同区域的线程聚合以用于执行，可将局部高速缓存没有命中或者局部高速缓存冲出的效果最小化。

[0017] 根据本发明的第五方面，提供了一种计算机程序，其当被数字逻辑执行时，执行如权利要求 1 所述的方法。还提供了一种计算机可读介质，其包含所述计算机程序。

[0018] 当利用这里描述的教导时，处理器元件、处理资源、核以及处理器被解释为等同的。处理器元件的繁忙状态可等同于其当前工作负载。在其所附从属权利要求中定义了另外的有利特性。

[0019] 根据本发明的另一方面，提供了一种多核处理器，所述多核处理器包括 :多个处理器元件 ;至少一个分配队列，所述分配队列按对执行的适任性的顺序列出可执行事务 ;以及多级调度器，其包括多个个体可执行事务调度器，其中每个个体可执行事务调度器包括调度算法，所述调度算法用于从准备好执行的多个候选可执行事务中确定用于执行的最适任可执行事务 ;其中，将多个调度器链接在一起并将所述多级调度器布置为从其将所述最适任可执行事务输出到所述至少一个分配队列。

[0020] 可执行事务可包括线程描述符，线程描述符可从多个状态中选择。根据状态转换配置，线程描述符可在来自所述多个状态的状态之间变化，由此识别所述可执行事务，从而可在多个可执行事务之间对其进行管理，以提供低调度等待时间以及调度层级的完整性。调度器可识别线程描述符。调度器可包括从多个调度状态中选择的调度器状态。控制调度器状态来支持动态调度层级，其中，可在系统的正常运行期间调整调度层级，同时保持在该层级内调度的项目的顺序和完整性。

[0021] 多核处理器还可包括硬件定时资源，可以布置硬件定时资源来提供 watchdog(看门狗) 超时，其中，watchdog 超时指的是已进入不可操作状态的处理资源实例。另选地，硬

件定时资源可提供时间片 (timeslice) 超时，其中时间片超时指的是在多个同等适任的可执行事务之间公平地共享的处理资源实例或处理资源实例的组。公平共享可包括提供对时间的同等共享或对与可执行事务的要求成比例的时间的共享。可将该硬件定时资源布置为在第一模式和第二模式之间切换，在所述第一模式下将其配置为提供 watchdog 超时，在所述第二模式下将其配置为提供时间片超时。优选地将硬件定时资源配置为如果设置了时间片断超时，就切换到第一模式。

[0022] 所述多核处理器的等待管理器 (pending manager) 还可包括定时器队列，每个定时器队列布置为接收定时器队列元素。定时器队列元素可包括可执行事务。定时器队列中的第一个可执行事务可以与第一时间参数相关联。第一时间参数指的是超时时间，所述超时时间是相关联的可执行事务应该变得对执行适任时的时间。优选地，第一个可执行事务的超时时间与当前时间最接近。第二个可执行事务可以与第二时间参数相关联。第二时间参数指的是第二个可执行事务的超时时间与第一个可执行事务的超时时间之间的差。第三个可执行事务可以与第三时间参数相关联。第三时间参数指的是第三个可执行事务的超时时间与第二个可执行事务的超时时间之间的差。

[0023] 队列中的第一个可执行事务可与一时间参数相关联。该时间参数指的是相关联的可执行事务的超时与也在队列中的第二个可执行事务的超时之间的差。优选地，所述第二个可执行事务是队列中具有在所述第一个可执行事务的超时之前发生且与其最接近的超时的可执行事务。

[0024] 多核处理器还可包括多个派出队列。优选地，派出队列被布置为识别另外的派出队列。每个派出队列可包括派出队列描述符。这使得可具有灵活数量的被服务处理资源实例，并还使派出队列描述符能够被按顺序地询问。还可将派出队列布置为从当前正使用处理器元件的抢占可执行事务的组中识别可执行事务。还可将派出队列布置为从将随后使用处理器元件的抢占可执行事务的组中识别另外的可执行事务。从而派出队列保持由调度管理器所进行的最近一次调度判定的索引。

[0025] 为提供多级调度器，多个个体可执行事务调度器的链接定义了调度层级，其中每个可执行事务调度器具有相关联的调度层。可将可执行事务调度器配置为识别先前调度了可执行事务的可执行事务调度器。可选地，所述可执行事务调度器可识别所述可执行事务是否源自与处理器元件相关联的分配队列。所述可执行事务调度器可识别所述可执行事务是否来自抢占可执行事务的组。针对调度事件是“推送”事件的情况可使处理优化。当由可执行事务调度器调度可执行事务时，还可将所述可执行事务调度器配置为将校正参数发送到先前调度过该可执行事务的各个可执行事务调度器。所述校正参数使调度判定可以传播并使得可以保持多级调度器内的计数器的完整性。

[0026] 还提供了一种操作多核处理器系统的方法，该方法包括：设置客户机；以及为该客户机选择交互状态。所述交互状态可包括：空闲状态，在其期间客户机可被配置为在功率管理模式下工作；以及用户状态，在其期间客户机被配置为在用户或正常模式下执行可执行事务。优选地，所述交互状态还可包括 API 交互状态，在 API 交互状态期间，客户机被配置为在特权状态中执行可执行事务。可选地，所述交互状态可包括客户机中介状态 (client shim state)，在客户机中介状态期间，客户机可被配置为准备可执行事务的环境。优选地，所述方法还包括设置服务器，其中，在所述客户机和服务器之间共享所述交互状态。优选

地，可提供带外信号以使所述交互状态改变。所述服务器可提供该带外信号。可选地，可执行事务可使所述交互状态改变。

具体实施方式

[0027] 介绍

[0028] 图 1 示出了与用于在多核系统中管理任务或线程的 SystemWeaver 中所实现的任务状态图相似的任务状态图。

[0029] 在多核系统中，调度器根据一组预定规则在最佳时间将工作包提供给最佳资源（所述的“调度”）。

[0030] 应用和下层的硬件平台都需要调度：

[0031] - 应用调度包括同步和适任性 (eligibility)。

[0032] 同步确保可以共享系统内的资源，而不会危及数据的完整性以及无论如何也不会危及整个系统。适任性确保以与应用的需要（其是在调度策略中表达的）相符合的方式将就绪任务分发给处理资源。

[0033] - 基于平台 / 分配调度定义了这样的策略，即，在适当的处理资源实例之间分配应用任务。这意味着在多个用户和 / 或多个不同的算法之间共享处理资源。

[0034] 图 2 示出了调度点相对于其造成的队列点的表示。

[0035] 从左至右遇到的第一个队列点是待定 (pending) 队列。被阻塞的任务根据其优先级存储在该待定队列中并由同步事件释放，进一步讨论待定队列的结构和行为超出了本文档的范围。第二队列点是就绪队列，其包括应用调度和分配调度。在图 2 中其被分为三个逻辑部分；应用调度和分配调度。在概念上，在这两个调度级之间是这样的点，在该点处所有当前就绪的应用任务都已根据其适任性（其是在用户定义的量度和调度策略中表达的）而排序。这是被称作分配节点的点。

[0036] 在分配节点之前，配置了一组应用特定调度策略，其确定了不同类别的任务以及同一类别的任务实例是如何竞争以访问处理资源实例的。这种调度策略的层级被称为调度锥 (cone) 并且是应用特定的。

[0037] 在调度锥之后，配置了一组平台特定调度策略，其确定了如何将最适任的应用任务分配给存在于下层硬件平台中的处理资源实例。这种调度策略的层级被称为分配锥并且是平台特定的。

[0038] 可以基于许多属性的组合来判断调度策略和实现的有效性：

[0039] - 吞吐量：每秒可进行的调度判定的数量。

[0040] - 等待时间：在系统内的事件与完成与该事件相关联的调度操作之间经过的时间。

[0041] - 可预测性 / 确定性：确定系统在所有情形下如何运转的能力。

[0042] - 效率：其中可实现任何给定调度算法的效率。这可以根据每次判定的指令（指令集效率的度量）和 / 或硅足印 (silicon footprint)（存储器和其他模区域）来测量。

[0043] - 策略：支持策略的多样性和将它们组合以形成复杂层级的能力。

[0044] 组件

[0045] SystemWeaver 有两个主要组件：服务器核和客户机中介 (shims)。可以以各种方

式将它们连接起来。支持 SystemWeaver 的系统将包括服务器核以及至少一个客户机中介。

[0046] SystemWeaver 核

[0047] SystemWeaver 核包括硬件引擎和紧密耦合的存储器。该存储器包含调度配置以及用于表示系统内的工作单位的动态描述符。每个 SystemWeaver 核跨多个客户机聚合，所述多个客户机可以为基于指令集的架构或硬件加速器。SystemWeaver 通过两个逻辑分立的数据路径与每个客户机单独通信：

[0048] - 带外信号，用于警告客户机需要注意的系统状态的变化。

[0049] SystemWeaver 核是该接口的主设备，（这里以及以后假设）该接口被典型地实现为中断。

[0050] - 数据路径，客户机可以通过其询问 SystemWeaver。客户机是该接口的主设备，该接口可被实现为总线、双工串行接口或者任何其他双向的实现。

[0051] 在引导过程中必须初始化 SystemWeaver 核。通常将会指派一个客户机代表系统的其他部分作为引导主设备并对 SystemWeaver 及其关联存储器进行初始化。

[0052] SystemWeaver 客户机中介

[0053] 在经典的构造中，每个客户机具有单独的客户机中介，然而更保守的实现可以在多个客户机之上聚合客户机中介。可以以硬件或软件来实现客户机中介。图 3 示出了软件客户机中介的主要要素：

[0054] -SystemWeaver HAL 实现 SystemWeaver 核的寄存器接口所需的命令格式化。

[0055] -SystemWeaver API 使应用能够以基于任务的抽象概念对 SystemWeaver 核进行调用。

[0056] - 用户线程是使用 SystemWeaver 任务管理能力的应用线程。在任何时刻，各个单独的客户机仅仅直接管理一个用户线程。

[0057] - 客户机中介对带外信号业务（通常中断业务）进行处理。处理资源实例内由客户机中介管理的环境主要确保：通过任务切换和抢占而保持完整性。通常，客户机中介包含架构不可知的部分以及指令集架构特定部分。

[0058] - 空闲代理执行管理任务，该管理任务处理个体处理资源实例或宏架构的断电 (power-down) 模式。

[0059] 图 4 示出了运转中的 SystemWeaver 的状态图。客户机中介执行两个主要功能：

[0060] - 对在其中执行处理资源实例的“环境”（对于经典处理器，环境可包含处理器栈空间和内容、寄存器值、程序计数器等）的管理（在适当时间分配、存储和恢复）。存在两种类型的环境：用户或任务环境，其是其中执行用户任务的环境；和处理实例特定环境，其是专门用于客户机中介管理操作的环境。

[0061] - 对操作模式的管理（在经典处理器中，用户（正常）和超级用户（有特权）模式可用，其定义了访问特定关键系统资源的权限，例如，与超级用户模式任务不同，将不允许用户模式任务访问会负面影响共享处理资源的其他用户模式任务的资源）。

[0062] 以下描述针对的是通用处理器，但是对于所有客户机类型存在类似性。

[0063] -“空闲”状态。在空闲状态期间，用户定义的算法可以利用由端处理资源实例（例如，时钟选通或其他低功率状态）或整个系统架构（例如，时钟选通、或者降低或消除向特定处理资源实例提供的电力）使其可用的断电模式。当在该状态中时，处理资源实例可以

在特权模式下工作并将使用处理资源特定环境。

[0064] 要注意，SystemWeaver 服务器没有指示客户机进入空闲状态，客户机之所以这样做是由于没有调度的任务。

[0065] 每个客户机保持在“空闲”状态中，直到来自 SystemWeaver 服务器的带外信号（通常为中断）指示其进行响应。

[0066] -“客户机中介”状态。客户机中介状态管理用户任务和空闲任务的执行环境。当在“客户机中介”状态中时，客户机中介保存已完成执行、被抢占或已阻塞的任何任务的环境，并为下一执行的任务恢复或创建环境（在空闲状态的情况下，其是处理资源实例特定环境）。当在该状态中时，处理资源实例可以在特权模式下工作。有时，客户机中介可以在处理资源特定环境或者用户或任务环境中工作。

[0067] 由于来自 SystemWeaver 服务器的带外信号（从“用户”或“空闲”状态转变到“客户机中介”状态）或者由于使执行中任务被阻塞的 SystemWeaver API 调用（例如由于尝试锁住信号机 (semaphore) 失败而引起从“SyWAPI”状态转变到“客户机中介”状态）而进入“客户机中介”状态。

[0068] 当完成了处理时，不用任何其他的外部信令发送，客户机中介可以从“客户机中介状态”转变到“空闲”状态（如果对于该处理资源实例不存在未完结任务）或者转变到“用户”状态（如果对于该处理资源实例存在适任任务）。

[0069] -“用户”状态。当在“用户”状态中时，客户机中介执行用户应用代码。在该状态中，处理资源实例通常将在“用户”或“正常”模式下工作。“用户”状态将完全在用户或任务环境中运转。

[0070] “用户”状态可以从“客户机中介”状态进入（由于开始或继续执行用户任务而引起）或者从 SyWAPI 状态进入（由于从 SystemWeaver 服务器 API 调用返回而引起）。

[0071] 由于任务完成或抢占（接收到从 SystemWeaver 服务器发送的带外信号），客户机中介会从“用户”状态转变到“客户机中介”状态。由于对 SystemWeaver 服务器 API 的调用，客户机中介会从“用户”状态转变到 SyWAPI 状态。

[0072] -“SyWAPI”状态。在用户任务需要与 SystemWeaver 核进行交互的情况下，用户任务通过使客户机中介状态变为“SyWAPI”的 SystemWeaver API 来这样做。当在该状态中时，处理资源实例可以在特权模式下工作。“SyWAPI”状态将完全在用户或任务环境中运转。

[0073] 在对 SystemWeaver API 进行调用之后进入“SyWAPI”状态。对于非阻塞调用，客户机中介将从“SyWAPI”状态返回到“用户”状态，然而，某些访问（例如与信号机相关联的访问）会使用户任务阻塞（阻塞的任务必须等待直到某些共享系统资源变为可用为止）。在此情况下，客户机中介转变到“客户机中介”状态。

[0074] 概念

[0075] 下面的部分讨论理解 SystemWeaver 调度器的操作所需的概念。

[0076] SystemWeaver 存储器元件

[0077] SystemWeaver 需要附加的紧密耦合存储器，使用该存储器来存储调度策略使得在整个系统开发过程中能够进行全面的调度修改以及能够调整灵活性。SystemWeaver 存储器被分为多个 SystemWeaver 存储器元件 (WME)。使用 WME 来表示下面讨论的任务描述符和调度描述符。

[0078] 任务描述符

[0079] 任务描述符是 SystemWeaver 架构的关键“流通单位”。它们表示根据在调度层级内配置的规则而竞争访问处理资源实例的工作单位。任务描述符包含：

[0080] -对任务控制块的引用，任务控制块进而包含对要执行的任务的引用以及该任务必须在其上执行的数据集。

[0081] -调度量度，定义任务的适任性。

[0082] -对于最初就会被阻塞的任务，还可包括同步基准和超时设定。对阻塞的任务行为的更详细的描述超出了本文档的范围。

[0083] -对“入口节点”的引用，其定义了调度层级的（可能在同步后）必须添加任务描述符的部分。

[0084] 调度锥和分配锥

[0085] 两种类型的锥用于描述 SystemWeaver 的调度行为：调度锥和分配锥。调度锥用于描述从很多“入口”点汇聚到单个聚合点（多对一映射）的调度器的层级。分配锥用于描述从单个聚合点分散到多个“派出 (dispatch)”点（一对多映射）的调度器的层级。

[0086] 调度锥

[0087] 调度锥（图 5 中红色所示）定义了“应用判定节点”层级，其由应用的需要所驱动（也如图 2 的就绪状态中的“应用调度”所示）。调度锥是多对一映射，其定义规则，根据这些规则，多个类别的任务以及任务类别的多个实例竞争系统资源。

[0088] 分配锥

[0089] 分配锥（图 5 中紫色所示）定义了“分配判定节点”层级，其主要由下层硬件平台的属性所驱动（也如图 2 的就绪状态中的“分配调度”所示）。分配锥定义规则，根据这些规则，在可用且适当的处理资源之间分配最适任的候选调度锥。

[0090] 主要调度节点

[0091] 存在三种用于描述调度配置的主要节点：入口节点、分配节点和派出节点。这些主要节点覆盖在下面的次要节点结构上，这更贴切地反映了调度器的详细实现。

[0092] 入口节点

[0093] 入口节点定义了新任务排队的点。通常，入口节点多对一地映射到分配节点上，作为调度锥的两个极端。入口节点可以与特定类别的任务相关联或者依照其他得自应用的策略。一给定入口节点仅可映射到单个分配节点上。

[0094] 分配节点

[0095] 分配节点定义调度锥和分配锥之间的勾画 (delineation)。分配节点通常代表一类处理资源。调度锥通常将一个或更多个入口节点映射到单个分配节点上，分配锥通常将单个分配节点映射到多个派出节点上，从而最终映射到处理资源实例。

[0096] 派出节点

[0097] 派出节点定义与个体处理资源实例相关联的出口点。派出节点通常与存在于硬件平台内的 IP 核一对一地映射（尽管可以向硬件多线程处理器核分配多个派出队列）。多个分配锥可映射到个体派出节点上。

[0098] 次要调度节点

[0099] 定义了两种类型的判定节点：应用判定节点和分配判定节点。虽然应用判定节点

和分配判定节点直接映射到调度层上，但是它们没有穷尽地定义调度器实现内的下层的数量或类型。

[0100] 应用判定节点

[0101] 应用判定节点定义调度锥内的中间调度或聚合点。每个应用判定节点定义一规则，可以根据该规则选择一组最适任的候选项。

[0102] 分配判定节点

[0103] 在多个分配节点映射到单个派出节点的情况下，要求分配判定节点设置这样的策略，该策略确定可以访问处理资源实例的分配锥。

[0104] 调度配置

[0105] 就绪状态结构（图 2）包含准备好执行的线程。整个就绪状态结构可包括许多调度锥和分配锥。这些线程是用独立的线程原语 (primitive) 创建的（即，这些线程是在就绪状态中创建的），或者这些线程接收到它们所依赖的同步原语或超时设定。经同步的线程是先前从阻塞状态转变的。

[0106] 就绪状态结构可包含调度器节点描述符和独立的线程描述符。虽然在运行期间允许线程描述符及其相关联的动态调度器层描述符来来往往，但是主要在系统初始化期间定义该结构。

[0107] 就绪状态结构使得可将线程调度到处理节点池或一特定处理节点。这使得能够实现负载平衡或跨多个兼容处理资源的其他分配行为，同时保持在有特定能力的处理资源（例如硬件加速器或 IO 装置）处发现特定任务的能力。

[0108] 调度层是用于实现构成就绪状态结构的主要和次要调度节点的原语资源。调度层可具有与其他调度层和任务描述符的父、子或同辈关系。

[0109] 图 6 示出了展示父子关系的调度器层级图。在该示例中，y 是 a、b 和 c 的父亲。y 是 x 和 z 的同辈。父亲仅可是调度层，而孩子可以是调度层或任务描述符。一给定同辈组（例如 a、b 和 c）可包括混合的任务描述符和调度层。进而，所有调度层都有父亲（派出节点是不定义父亲的唯一描述符）。

[0110] 在运行期间，父亲层可根据用户定义的策略从最适任的孩子继承“量度”（优先级等）。在较深嵌入的调度策略需要从正被比较的调度分支了解适任候选项的情况下可以使用该特性（在下面的量度传播章节中详细说明了该主题）。

[0111] 下面的章节描述任何 SystemWeaver 调度层级的组成部分。

[0112] 基本调度器层

[0113] 调度器层定义了用于调度线程描述符的层级。每个调度器层通常定义：调度算法、用于确定调度判定的某些量度、用于定义如何将这些量度从孩子传播到父亲的继承策略、以及可以为其他调度器层或线程描述符的孩子元素的列表。存在三种类型的调度器层描述符：根、静态和动态（其中动态层是一特殊类型的静态调度层）。图 7 示出了一示例调度锥的实现。图 36 示出了从图 7 开始的所有调度图的图参考图例。

[0114] 调度器根描述符

[0115] 调度器根描述符与派出队列的一对一映射。调度器根描述符表示就绪状态结构中的最终节点。根描述符量度总是包含根据定义的继承策略从就绪状态结构得出的量度的副本。

[0116] 调度器根描述符是在系统初始化期间配置的，并且永久存在。

[0117] 静态调度器描述符

[0118] 在调度层级中，调度器静态描述符存在于根节点之下。静态调度器描述符的父亲可以是其他静态调度器描述符或根描述符。调度器静态描述符根据其父亲定义的调度器算法和其自身的调度器量度，与兄弟节点竞争。

[0119] 调度器静态描述符是在系统初始化期间配置的，并且永久存在。在操作期间，SystemWeaver 根据选定的调度以及量度传播算法来保持调度器量度。

[0120] 动态调度器描述符

[0121] 在调度层级中，调度器动态描述符存在于根节点之下，并且可选地存在于静态节点之下。动态调度器描述符的父亲可以是静态调度器描述符或根描述符。动态调度器描述符根据其父亲定义的调度器算法和其自身的调度器量度，与兄弟节点竞争。

[0122] 动态调度器描述符可以在任何时间配置。这使得与纯静态可能提供的相比，系统能够支持高得多的数量的调度层。SystemWeaver 是利用这样的可能性来实现此点的，即，虽然在总体时间上使用大量且不同的线程和动态调度器层，但是在有限时段内的瞬时需求较小。例如，在具有支持最大 4k 动态元素（线程和动态调度器描述符）的附加存储器的网络系统中，在任意时刻可以支持 16k 个连接，但是在处理器中仅有来自整个连接空间的一小部分的数据单元会激活。这种灵活性是以性能小幅下降为代价而实现的，因为，如果不存在动态调度器描述符，则必须在添加子线程描述符之前创建它。

[0123] 在操作期间，SystemWeaver 根据选定的调度算法保持调度器量度。在某些情况下，SystemWeaver 将把动态调度器描述符释放回 WME 空闲列表。

[0124] 处理器资源池

[0125] 处理器资源池使得能够将特定处理资源的实例聚合到单个分配节点。然后，该分配节点可提供跨处理资源池的个体成员的负载平衡、智能抢占和功率管理。

[0126] 图 8 示出了典型的处理资源池的配置。三种新定义的 Weaver 存储器元件支持处理器池的配置结构：

[0127] 池附加节点

[0128] 池附加节点 (PAN) 用于将调度器根层附加到处理资源池根层。PAN 必须存在于调度器根层内（即，其父亲必须是调度器根节点）。在操作期间，用池根节点 (PRN) 的量度的副本自动更新 PAN 量度，PRN 的量度是依次从调度锥继承的。

[0129] 没有使用在 PAN 内定义的调度操作符。

[0130] 池静态节点

[0131] 池静态节点 (PSN) 用于将调度器根层附加到处理资源池根层。PSN 存在于池根层内（即，其父亲必须是 PAN）并自动保持派出节点（即，当前在执行的线程）的量度的副本。

[0132] 必须将给定池的 PSN 内的调度器操作符全部设置为相同的算法，定义用于选择要抢占的适当处理资源实例的策略。

[0133] 池根节点

[0134] 对于每个处理资源池，存在单个池根节点 (PRN)。池根节点定义了处理资源池的分配节点。PRN 内的量度反映了与该分配节点相关联的调度锥内拥有的最适任线程。必须将 PRN 父指针设置为指向池静态节点之一。

[0135] 与正常情况一样，应该根据调度锥的相邻层的需要来设置调度器算法。

[0136] 动态调度器配置

[0137] SystemWeaver 支持在运行时创建和删除调度节点以及提供在无损失或不打乱顺序的情况下将任务类别从一个入口节点迁移到另一个入口节点的能力。当讨论动态调度器配置时必须引入两个另外的概念：休眠调度层和标记线程。

[0138] - 休眠调度层存在于层级内并且可以接受推送操作（即，可以累积子条目），但是不适任于调度，因而从不弹出。

[0139] - 标记线程仅当其为依赖于调度层级的特定部分的最后一个线程时才被调度。依赖于调度层级的一部分的线程数包括就绪线程数和在其变为就绪时会使用调度层级的该部分的阻塞线程数。标记线程可以像任何其他线程一样带有任务引用，并且通常用于完成对调度层级的一部分和另一部分之间的转变操作进行的管理。

[0140] 下面的章节详细描述来自层级的一部分的任务流的转变的示例序列。要注意，其是调度层级的一部分的删除的超集。

[0141] 操作序列

[0142] 较高级别的软件有责任确保遵守适当的操作顺序。无法遵守该顺序会导致意想不到的行为，具体地，一定不能将新线程引入已插入了标记线程的调度层级的一部分。

[0143] 在该示例序列中，假设任务流 t_{stream} 从调度层级 h_1 转变到新的调度层级 h_2。

[0144] - 创建休眠调度器层级 h_2。

[0145] - 将 t_{stream} 上的所有新任务描述符分配到 h_2。

[0146] - 将标记线程插入 h_1 中。

[0147] - 等待标记线程的出现。

[0148] - 唤醒休眠层级 h_2。

[0149] 调度器分析、算法、操作符和操作数

[0150] 调度分析有许多形式（EDF、RMA 等）并且通常为应用特定或者至少为扇区（sector）特定。调度分析的结果是一组策略，其静态或动态地控制应用的运行时间部署。通过其独特的宏架构，SystemWeaver 在运行时有效地执行这些预定义的策略 / 算法。

[0151] 调度算法

[0152] SystemWeaver 被设计为能够在硅设计时定义专用算法而不破坏架构或实现。然而，缺省提供了很多算法：

[0153] -FIFO 调度：简单的先入先出队列。

[0154] - 优先级调度：最适任的候选项具有最高（升序优先级）或最低（降序优先级）的优先级量度。

[0155] - 轮转：当从调度层级弹出一任务时，将调度判定更新为下一同等判定。要注意，轮转不是在调度层级的“最左”极端的相关调度策略。

[0156] - 加权公平队列：一种复杂的调度器，其中，根据所分配的权值和负载的某些测量值（即包的长度）来选择适任候选项。

[0157] 如果注意一下整个调度器层级，就可创建调度器算法的复杂组合，以在应用系统中提供精密的通信量和任务管理能力。

[0158] 操作符

[0159] 调度算法被进一步分解为单独的调度操作符和量度操作符，都在父节点中定义：

[0160] -调度操作符：定义了使用存储在子节点中的操作数来确定最适任候选项的方式。调度操作符不修改子节点中的操作数。

[0161] -量度操作符：定义了将最适任孩子的操作数传播为父亲的操作数的方式。传播操作符可以是空（对父亲不更新）、复制（覆写父亲的操作数）、或者包括针对一些和所有子操作数和父操作数的数学函数。在所有的情况下，子操作数都不改变。

[0162] 调度操作符和量度操作符本来就实现在 SystemWeaver 的调度器硬件中。调度操作符和量度操作符的组合通常用来定义给定的调度器算法。调度算法一般意味着对于推送事件（其中一新任务被推送进调度器层级）下的行为与弹出事件（其中从调度器层级弹出一任务）下的不同。例如，考虑 FIFO 调度器，当将一新任务推送进非空的 FIFO 调度级时，没有执行调度更新，而当从 FIFO 调度级弹出一项目时，必须更新调度器。

[0163] 调度操作符

[0164] 调度操作符被设计成可扩展的，但是定义了缺省操作符的选项。调度操作符通常是比较操作符，因而结果总是布尔值 (Boolean)。在下面的表中，M 根据以下方案表示调度层的成员或调度层描述符自身中的两个量度之一：

[0165] -$M_{current}n$ 是指属于当前最适任候选项的那些量度

[0166] -$M_{candidate}n$ 是指属于这样的候选项的那些量度，在调度更新的过程中当前描述符与所述候选项进行了比较

[0167] -$M_{tier}n$ 是指属于附有当前项和候选项的调度器层描述符的那些量度

[0168]

指令	调度器更新判定
GTR	$M_{current}0 > M_{candidate}0$
GTR_EQ	$M_{current}0 >= M_{candidate}0$
LESS	$M_{current}0 < M_{candidate}0$
LESS_EQ	$M_{current}0 <= M_{candidate}0$
EQ	$M_{current}0 == M_{candidate}0$
UPDATE_EXIT	TRUE（真）
UPDATE_NO_EXIT	TRUE
NO_UPDATE_EXIT	FALSE（假）
HYSTERESIS（滞后）	$(M_{current}1+M_{tier}1) < M_{candidate}1$

[0169] 表 1：调度操作符

[0170] 复合调度操作符

[0171] 还可使用复合调度操作符，其是表 1 中的调度操作符的组合。例如：

[0172] 需要更新＝ ($M_{current}0 > M_{candidate}0$)&&($M_{current}1 < M_{candidate}1$)

[0173] 其中，所涉及的参数可以利用父描述符和子描述符中的量度。可以将这些复合操作符用于传统调度层和池分配层两者中。

[0174] 对于进一步的信息和示例，请参照下面的调度序列图章节。

[0175] 量度操作符

[0176] 量度操作符在本质上是算术操作符。与调度操作符相同，量度操作符被设计成可扩展的，但是具有一组缺省操作符，见表 2。就复杂性而言，量度操作符包括从空或简单的复制操作一直到复杂的乘法累积操作。

[0177]

指令	调度的元素量度		旧量度内容		新量度内容	
	0	1	0	1	0	1
ALL	A	B	X	Y	A	B
NONE	A	B	X	Y	X	Y
PROP_0	A	B	X	Y	A	Y
PROP_1	A	B	X	Y	X	B
PROP_WFQ	A	B	X	Y	X+Y*B	Y
ADD	A	B	X	Y	X+A	Y
SUB	A	B	X	Y	X-A	Y

[0178] 表 2：量度操作符

[0179] 操作数

[0180] 调度操作数（或量度）分为两组：

[0181] 一局部量度与处理资源实例、调度器层以及线程描述符相关联。对局部量度进行操作会自动引起重新调度事件。

[0182] - 全局量度是可选的，并且通常与系统资源状态（诸如总线繁忙或空闲存储器的某些启发 (heuristic)）相关联。

[0183] 一给定调度算法可仅使用两个量度，其中的一个必须是局部的。通过 MetricIIsGlobal 标志来确定第二量度的类型：

[0184] - 当 MetricIIsGlobal 被重置时，量度 1 是局部的，并将在调度操作中被按原意 (literal) 使用。

[0185] - 当设置了 MetricIIsGlobal 时，量度 1 是对全局量度端口阵列的索引。

[0186] 局部量度

[0187] 任务描述符和调度层描述符都包含两个 32 位操作数或者调度量度。这些操作数在调度操作期间被它们各自的父亲使用并且可以在调度操作期间被转换和 / 或传播成父亲的操作数，以用于该层级中随后更高级别的调度。

[0188] 图 9 示出了存在于任务描述符内的调度量度。在任务描述符内，量度 0 通常用于表示任务的优先级。保留该量度的最低有效字节以供在 SystemWeaver 硬件和客户机软件内的内部使用。对于调度器层量度，没有这种限制。

[0189] 全局量度

[0190] 全局量度在本质上是被动的，全局量度值的改变不会引起针对所有潜在受影响的调度资源的重新调度事件。在这样的时刻（由于某些其他系统事件而调度了依赖的调度资源时）询问全局量度。虽然 SystemWeaver 架构没有对全局度量的使用强加限制，但是它们可用于系统启发（总线利用、在一时间窗口中的存储器饱和度 (fill) 等），因此变化率相对较低。也可应用过滤器以使数据平均。

[0191] 调度层级配置细节

[0192] 在以下章节提及的所有配置图都使用如图 36 所示的共同格式。

[0193] 图 10 示出了单个处理器的最基本的配置，其示出了这样的配置 :单个处理资源实例（单个派出节点）及其强制调度根节点。在该最简单的情况下，因为仅有单个处理器，所以调度器锥包括单个 FIFO 级并且分配级为空。因此，调度器根节点既是入口节点又是分配节点。

[0194] 要注意，调度器节点实现上的箭头是从右向左（父到子）地示出的，这与任务的“流”相反，任务“流”从孩子流向处理资源实例。

[0195] 调度器是以模块的形式实现的并配置在易失性存储器中，这使得可以根据不同策略的连续的调度器层来构建非常精密的调度器层级，并且可以在整个开发过程中对其进行调整和裁减。然而，可能有无效配置，必须特别当心以确保适当的量度对于深深嵌套的调度器层可用。

[0196] 层内结构

[0197] 调度层按到达的顺序或者根据缺省的 FIFO 排队策略来存储条目。稍后描述将所定义的调度策略覆盖在该结构上的方式。新节点（描述符）通过推送操作被添加到层内结构，并由于弹出操作而被移除。调度操作不操作层内链接。

[0198] 图 11 示出了单处理资源实例的更有代表性的调度结构。在该示例中，从左侧起，两个 FIFO 级供给一个优先级级。在层级的两个级处存在三个调度层。要注意，调度层仅具有一个“出口”节点（在该图的右侧示出），但是潜在地具有许多入口节点（在该图的左侧示出）。

[0199] 图 12 示出了 FIFO 调度层中的一层的详细图。该图示出了一组指针，该组指针保持该层上所有同辈之间的双链表。

[0200] 双链表用于使该层的任意成员的移除（弹出）的性能最大化。

[0201] 虽然该详细图仅示出了任务描述符，但是该结构可以同等地应用于包含线程和调度节点的任何混合的层。

[0202] 同辈元素之间的层内链接仅在推送和弹出操作期间被操作。

[0203] 层间结构

[0204] 除了池根层，图 12 示出了层间链路的结构。各层具有一父节点，该父节点必须是调度器根节点或者是调度节点。这些节点存储指向该层的最适任成员的指针。在接收到调度事件时，根据在父节点和子节点内分别定义的调度策略和量度来更新这些子指针。

[0205] 每个子节点还必须引用其父亲。

[0206] 池根层结构

[0207] 池根层结构是一特殊情况，在该情况下层具有单个入口节点和很多出口节点。该入口节点是调度锥向其汇聚的点（如图 2 中的就绪队列结构的“应用调度”部分所示），也被称为“分配节点”。“出口节点”将池根层链接到其上可能分配有任务的处理资源实例的“分配调度”结构。图 13 示出了包含两个处理资源实例的池的此类型的结构的表示。

[0208] 每个池分配层必须包含池根节点 (PRN) 以及一个或更多个池静态节点 (PSN)，不允许其它节点类型。PRN 包含对调度锥（存储在 HeadIndex 字段内）的第一层的引用以及对考虑分配的第一 PSN 条目的引用。必须将共同的分配和量度更新策略分别存储在每个 PSN 的调度器和量度的推送和弹出操作符内。

[0209] 每个 PSN 必须将 PRN 作为其孩子引用（使用 HeadIndex 字段）。

[0210] 池静态节点的父亲必须是池附加节点 (PAN)。PAN 和 PSN 必须具有一对一的映射。然而，各处理资源实例可具有与其所参与的各个分配池相关联的多个 PAN。图 14 示出了处理资源参与两个池的情况下的示例结构。对其中给定处理资源可以作为成员的池的数量没有进行限制。此外，任何池可以和任意数量的其他池共享任意数量的作为其组成部分的处理资源。

[0211] 在调度根节点内，存在与处理资源实例所参与的各个分配锥相关联的两个 PAN。此外，存在一调度节点，其提供对所需处理资源实例的特定访问。

[0212] 在池分配层的各 PSN 内定义的分配策略识别选择用于执行给定任务的最适任处理资源实例的方式。例如，该策略可以是优先级之一，其中，在来自相关联的调度锥的高优先级任务到达时，选择当前在执行的最低优先级任务的处理资源实例用于抢占。

[0213] 图 15 示出了包括五个处理资源实例和两个分配池的配置。要注意，PRI#3 参与两个池。

[0214] 行为

[0215] 下面的章节描述 SystemWeaver 调度的行为。

[0216] 一般原理

[0217] 下面的章节提供用于解释 SystemWeaver 调度架构的某些关键基础原理的某些基本背景信息。

[0218] 基于指针的队列

[0219] 虽然在 SystemWeaver 内存在多个潜在的排队点（图 2 中已详细描述），但是这些点是仅使用指针而实现的。从未复制排队的实体（即 SystemWeaver 存储器元件 (WME)）。

[0220] 基于事件的调度

[0221] SystemWeaver 仅当系统状态的某些改变要求更新调度判定时才这样做。这些状态的改变可被分为三个事件类别：

[0222] -“推送事件”，其中，系统状态的改变使得将新线程描述符引入就绪状态结构（要注意，其可以是新线程描述符，或者可以是系统状态的改变使其变为就绪的线程的现存线程描述符）。

[0223] -“弹出事件”，其中，系统状态的改变导致从就绪队列结构中移除线程描述符。

[0224] -“更新事件”，其中，调度参数已被修改，需要重新评估调度判定。

[0225] 这些改变可以是：

[0226] - 中断（“推送事件”，因为与该中断相关联的阻塞线程移至就绪状态）。

[0227] - 由执行中任务创建的新任务到达（如果新任务不依赖于其他因素或事件，则其可以是推送事件，）

[0228] - 同步事件，例如信号机信号（假设有一阻塞的线程在等待该信号，则其是“推送事件”，因为该阻塞线程的描述符转变到就绪状态）。

[0229] - 任务的执行“优先级”的改变，即“更新事件”。

[0230] - 处理资源实例内的任务的消耗（从就绪转变到执行）（“弹出事件”）。

[0231] - 任务的调度量度的修改（“更新事件”）。

[0232] - 调度器层的调度算法或量度的修改（“更新事件”）。

[0233] - 调度器层级自身的修改（“更新事件”）。

[0234] 在系统处于稳定状态的情况下，SystemWeaver 保持空闲。原则上，在最节能的方案中，这可以使 SystemWeaver 能够在需要附加调度的事件到达之前断电。要注意，改变全局量度不会引起重新调度事件。

[0235] “准时（just in time）”调度

[0236] 仅将排队到给定调度层的新条目与当前最适任条目（由父 HeadIndex 识别）进行比较。如果根据层调度策略，新条目比当前的头更为适任，则更新 HeadIndex 字段以引用该新条目。总是将新条目放置在当前链表结构的后部。

[0237] 如果调度策略是 FIFO，则在新条目到达时从不更新 HeadIndex 指针，除非该队列为空。因此，缺省行为（其中将新条目放置在队列的后部）与 FIFO 算法相同。

[0238] 该方案确保花费最少的时间处理推送操作，这一般是作为调度性能中的等待时间而被遵守的。因此，弹出调度更为繁重，在最差情况下，必须针对每次弹出操作评估调度层的全部内容以更新调度判定。然而，在物理队列结构中总是使用自然的 FIFO 算法是可取的，这是因为对调度算法的修改不需要将调度器层重新链接。此外，通常，弹出调度可与应用的执行并行地执行，因而对整体系统性能具有较小的影响。

[0239] 调度分析、策略和操作符

[0240] 存在许多对系统进行分析以确保满足实时绝限（deadline）的方法，其示例为 EDF（最早绝限优先）、RMS（速率单调调度）和各种其他随机方法。这些方法趋向为应用特定的，并且可能在本质上是私有的。然而，在所有情况下，这种调度分析的结果是一组调度策略（即，优先级、FIFO、轮转、加权公平队列），必须在运行时对其进行有效的部署。SystemWeaver 技术的目标是在运行时有效地执行通过调度分析而识别的策略。对于 SystemWeaver 内的部署，每个调度策略还要进一步解码为一组调度操作符。

[0241] 每个调度器层具有两个操作符，这两个操作符用于确定：由于推送至调度层（或从属调度层）或者从调度层（或从属调度层）弹出，如何更新调度判定。在某些情况下，调度操作符将需要操作数，所述操作数存储在调度器和任务描述符等的量度字段内。

[0242] 调度量度和量度传播操作符

[0243] 调度量度存储选定的调度算法可能需要的信息，其最基本的示例是优先级。在某些情况下，有必要将量度从最适任候选项转发到父节点，以使该信息可以直接在随后的调度判定中使用或者用作量度更新操作中的操作数。量度传播操作符定义了针对推送和弹出

的情形如何实现此点。

[0244] 根据调度节点在层级内的位置，量度字段还可反映当前正在执行的线程对给定处理资源的优先级。在此情况下，使用量度字段来确定是否需要抢占（见下面的调度行为章节）。

[0245] 调度资源

[0246] 下面的章节描述用于在运行时实现调度算法的各种资源。

[0247] 层

[0248] 调度器层

[0249] 调度器层包括一个父亲和多个孩子，该父亲可以是调度器根节点、池根节点或基本调度节点（图6）。孩子可以是基本调度节点、线程或任务描述符或池附加节点。通过使子节点能够凭本身的权利而成为调度节点（即，其他调度器层的父节点），可以建立复杂的调度器层级。

[0250] 池分配层

[0251] 池分配层可包含唯一的池根节点（仅一个）以及多个池静态节点。每个处理类别仅有一个池根节点。

[0252] 派出队列描述符

[0253] - 调度器操作符：用于定义确定当前在执行的任务是否应被抢占的调度策略。

[0254] - 量度传播操作符：在派出队列描述符中没有量度传播操作符。

[0255] - 量度：量度元素通常存储当前在执行的线程的量度。

[0256] 调度器和池根节点

[0257] - 调度器操作符：用于确定调度锥的最适任候选项。

[0258] - 量度传播操作符：其总是被设置为继承调度锥的最适任候选项的量度。

[0259] - 量度：保持调度锥的当前最适任候选项的量度。

[0260] 调度器层元素

[0261] - 调度器操作符：用于确定来自附加层的最适任孩子候选。

[0262] - 量度传播操作符：用户定义。根据后续调度级的需要进行设置。

[0263] - 量度：用户定义。根据后续调度级的需要进行设置。要注意，某些量度传播操作符会自动更新这些字段。

[0264] 池静态节点

[0265] - 调度器操作符：用于确定池分配层中用于抢占的最适任候选。

[0266] - 量度传播操作符：用于确定执行中的任务的量度的传播。

[0267] - 量度：根据池分配算法的需要进行设置。缺省地，所述量度将反映当前在执行的线程的量度，然而对于某些分配策略，可能需要静态分配。

[0268] 池附加节点

[0269] - 调度器操作符：未使用。

[0270] - 量度传播操作符：用于控制最适任任务量度的传播。

[0271] - 量度：用于存储附加到关联池根节点的调度锥的最适任任务的量度。

[0272] 线程元素

[0273] - 量度：用于传达与用于调度的任务的适任性直接相关的信息，或者根据所述量

度，调度器可以计算出该适任性。

[0274] 调度行为

[0275] 调度操作分为两个子类别：

[0276] -标准层调度，其中，调度器层内的一个或更多个条目竞争成为层内的最适任条目。

[0277] -池分配调度：识别应该中断处理资源实例中的哪个选定实例。

[0278] 除非接收到调度事件，否则不发生调度行为。

[0279] 调度推送和弹出事件

[0280] 如前所述，系统状态的改变会引起“推送事件”或“弹出事件”，这些事件引起重新调度的发生。所有调度操作都是工作守恒的(work conserving)。仅对调度层级中被给定事件确信无疑地影响到的部分进行重新评估，称这些部分存在于重新调度范围内。图17示出了对基本调度层级上的事件的重新调度范围，图18示出了简单的两个实例的处理资源池的重新调度范围。

[0281] 层调度

[0282] 层调度是SystemWeaver调度算法的最基本的构建块。调度事件可以引起可由用户配置的调度层级所定义的连续的层调度操作。各个层调度操作的结果是更新了父调度器（调度节点或调度器根节点）的HeadIndex指针。父调度器的量度还可以根据所定义的量度传播算法进行更新。

[0283] 原则上，层调度在当前HeadIndex处开始并对调度器层的成员进行叠代（但是在实践中，为了使等待时间最小，推送操作仅仅更新针对当前头指针的调度判定），根据以下内容确立是否需要更新HeadIndex：

[0284] -事件，其可以是推送或弹出操作

[0285] -与事件类型（推送或弹出）相关联的调度算法

[0286] -层成员的量度

[0287] 如果找到更适任的条目，则相应地更新HeadIndex。观察了几种精练了调度操作的行为的特殊情况。在所有情况下，在调度操作中忽略休眠调度器层。

[0288] 各个调度节点必须总是知晓存在于其子层级中的线程或任务描述符的数量，以确保保持了关键饱和度参数。然而，不必总是完全地调度每个层，保持一标志，该标志识别直接下游的重新调度操作在哪里引起了调度判定更新，如果一层引起了调度判定更新，则父层也必须被完全评估；如果没有，则不需要对剩余的上游调度器层级进行重新调度（但是需要更新某些其他状态）。

[0289] 任何重新调度的最后一个操作都是确定是否应该允许最适任就绪任务抢占当前在给定PRI上执行的任务。派出队列描述符包含调度算法和当前运行的任务的量度两者——可以针对调度器根节点量度对这些量度进行评估，所述调度器根节点量度包含来自调度锥的最适任线程量度的副本。

[0290] 池分配调度

[0291] 池分配调度仅发生在池分配层内。而基本层调度寻求找到用于执行的最适任线程/任务的候选，池分配调度寻求找到用于抢占的最适任处理资源实例候选。通常，这意味着识别以资源池中的最低适任性运行任务的处理资源实例，并且将其与来自附加调度锥的最

适任“就绪”任务的量度进行比较。

[0292] 在最适任就绪任务的适任性比所有运行中的任务都低的情况下，对每个附加处理资源实例上的剩余的分配锥进行更新，以确保所有调度层中的每一个保持知晓可访问的下游任务的总数，然而，不必进一步调度。

[0293] 在识别出抢占候选项的情况下，调度更新仅向所述处理资源实例传播。

[0294] 图 20 示出了由于图 18 所示的推送事件而发生的重新调度的序列图。在池根层（层 #6）发生基本层调度操作，接下来是池分配调度操作。在此示例中，将节点 5 选为适任于抢占，因此执行层 #1 中的层调度操作。随后的派出层调度操作导致对附加处理资源实例的抢占。之后，还更新层 #2 以确保可以保持其下游任务 / 线程的计数值。

[0295] 协作和抢占调度

[0296] 抢占调度使得当前在执行的任务能够被更适任（更高优先级）的任务异步地中断。抢占产生对执行处理资源和环境（例如，存储状态并且抢占者一离开该资源就继续执行的能力）的特定需求。通常，可抢占的任务或线程在就绪和执行状态中会保持相同的调度适任性。

[0297] 相反，协作线程仅在完成时退让，因此更高优先级的任务必须等待。在 SystemWeaver 的任务管理方案中，协作线程在其进入执行状态时使其适任性最大，从而阻止出现更高优先级的任务以及潜在的随后抢占。

[0298] 量度传播行为

[0299] 调度事件或执行中任务或线程的量度的修改都会引起量度传播。

[0300] 调度事件量度传播

[0301] 当由于调度事件而更新了父亲的用于调度层的 HeadIndex 时，根据在父层中所定义的量度传播操作符，将量度从最适任子量度传播进父量度。这依赖于操作的本质（推送或弹出事件），并且在复杂性上从简单复制到乘法累积。

[0302] 执行线程量度传播

[0303] 可以动态地修改当前在执行的线程的量度，这可用来消除对加锁资源的优先级倒置条件。在执行中的处理资源实例没有参与分配锥的情况下，仅更新派出队列描述符的量度。在分配池的情况下，将执行中的量度传播给与处理资源实例相关联的池静态节点（图 21）。由 PSN 自身内所保持的量度传播操作符来控制对 PSN 量度的更新。在特定调度情形中，静态值在池静态节点内必须持续存在。

[0304] 在两种情况下都引发重新调度事件，以确保新的执行中量度不会引起执行中任务和就绪的任务的相对适任性的改变。在非池的情况下，简单地相对于新的执行中量度对调度器根节点量度进行重新调度。在形成池的情况下，必须重新评估池分配层以及所有后续层。

[0305] 空闲处理

[0306] 当处理资源实例进入空闲状态时，其使用执行中量度来通知调度结构。实质上，空闲处理资源实例是“正执行”可能的最低优先级任务的实例，并因而会被任何到达的任务所抢占。将执行中量度设置为空闲值会以通常的方式引发重新调度事件，从而使该空闲任务被在就绪状态结构中等待该处理资源实例的任务所“抢占”。

[0307] 对于“空闲任务”的更详细的描述及其对处理资源池中的功率管理的影响，请参见

下面的池情形章节中的功率管理章节。

[0308] 高级调度模式

[0309] 固有地或者通过采用特定的 SystemWeaver 配置可以使用几种高级模式和行为。在下面的章节中描述以下模式。

[0310] 要注意，这不是在穷尽性地列举 SystemWeaver 内可用的调度模式。

[0311] 时间片

[0312] 虽然 SystemWeaver 系统主要是事件驱动的，但是基于传统定时器的系统（诸如时间分片）仍然可用。时间片型任务根据单独的时间片周期共享处理资源，所述时间片周期确定在其期间任务可占用处理资源的时间间隔（假设在该时间间隔期间没有优先的任务就绪）。

[0313] 相对于普通任务（图 1 中示出），时间片型任务展现出轻微改动的“运行”行为。图 22 示出了运行状态的两个部分：正常和受轻视 (deprecated)。

[0314] 此章节描述时间片任务的行为以及当对其进行配置时必须遵守的规则。

[0315] SystemWeaver 核资源

[0316] 下面的章节讨论用于实现 SystemWeaver 服务器核内的时间片特性的资源。

[0317] 计数器

[0318] 在 SystemWeaver 核内使用每处理资源实例计数器来推动时间片行为。还提供了单个预置器 (prescaler)，其由系统时钟提供。在芯片设计时设置该预置器的位分辨率。

[0319] 时间片状态指示器

[0320] 留出预处理资源实例中断状态寄存器内的状态位用于时间片行为。该状态位寄存时间片计数器的期限，并可被软件用来确定是否发生了时间片事件。

[0321] 配置

[0322] 时间片组内的所有任务必须共享相同的优先级和相同的父调度层，此外，时间片任务不应与其他非时间片任务共享调度层。应将时间片父亲的调度算法设置为 FIFO。图 23 示出了典型的时间片配置，其中时间片组在具有这样的一组前景的背景下工作，即，当其需要服务时采用优先级的事件驱动的任务。

[0323] 行为

[0324] 当时间片任务首先开始执行时，系统范围的时间片值被复制到与处理资源实例相关联的时间片计数器。称时间片任务进入其“正常”运行状态（图 24）。在正常状态下，每个循环将该计数器递减。当到达 0 时，硬件将任务的执行优先级（存储在派出队列描述符内）自动递减，从而任务进入“受轻视”状态。此时，时间片间隔计数器切换到传统的看门狗 (watchdog) 模式。

[0325] 在单个处理资源实例向多个时间片任务提供服务的情况下，（以关联的重新调度操作）使执行优先级递减的行动将引起被就绪状态结构内的时间片组的另一成员的抢占。通过询问时间片状态位，软件客户机可确定时间片的持续时间已届满，并且将该现在已被抢占的任务推送至 FIFO 队列的后部。因此，该组遵守所配置的时间片规则，同时很大程度上保持了 SystemWeaver 核调度以及客户机行为的正常运行模式。

[0326] 当在“正常的”时间片状态下时间片任务被非时间片任务抢占时，将未完结的时间片持续时间复制到任务控制块。然后将该任务推送回时间片组使其位于 FIFO 队列的头部。

当任何抢占任务处理完成时，所述时间片任务恢复，被抢占任务的剩余时间片被设置到时间片计数器。

[0327] 当时间片任务在“受轻视”状态下被抢占时，将其推送回时间片组 FIFO 队列的尾部。在两种情况下，时间片任务的优先级量度都保持为其原始配置值。

[0328] 在时间片组由处理资源实例的池提供服务（假设没有抢占任务）的情况下，进入“受轻视”状态不一定会立即切换到另一时间片组成员。要注意下面的观察：

[0329] $T_p = (t*d)/p$ 其中 $(1 \leqslant p \leqslant t)$

[0330] $T_{ready} = T_p - d$

[0331] T_p 时间片组完全循环一次的时长（执行每个

[0332] 成员任务一次）。

[0333] T_{ready} 每个循环中给定任务在就绪状态冲等待

[0334] 的时间的量。

[0335] t 时间片任务的数量

[0336] p 池中的处理资源实例的数量

[0337] d 每个时间片间隔的持续时间

[0338] 要注意，当 $p = t$ 时，在没有其他任何抢占任务时，时间片任务连续运行。

[0339] 执行概况

[0340] 下面的执行概况示出了 SystemWeaver 的时间片行为：

[0341] 图 25 示出了单处理器和三个时间片任务的经典情况。在抢占任务到达前，每个时间片任务根据时间片间隔进行时间共享，在抢占任务到达时，时间片任务（在此情况下为 2 个）退让。当高优先级的抢占任务完成时，原来的时间片任务继续执行以完成被中断的时间间隔。

[0342] 图 26 示出了具有两个处理器的同样情形。最初，在这两个处理器之间共享这三个时间片任务。当抢占任务到达时，这些时间片任务共享剩余的处理器，并且当抢占任务完成时，这些时间片任务在两个处理器上继续执行。

[0343] 图 27 展示了：当可用处理资源的数量等于时间片任务的数量时，各个时间片任务在处理器之一上连续运行。当高优先级任务取得对一个处理器的控制时，时间片组自动地根据所定义的时间片间隔共享剩余的处理器。

[0344] 池情形下的功率管理

[0345] 处理资源池的缺省行为是从分配层内的第一个池静态节点评估调度判定。池根节点 (PRN) 具有父指针，其通常指向分配层中的第一个池静态节点 (PSN)（图 28）。当评估用于抢占的候选项时，使用同辈指针从此条目开始比较并且围绕该链表进展。

[0346] 如果所有静态节点都具有相同的适任性并且该适任性低于来自调度锥的候选，则将选择遇到的第一个节点。因此，在低负载的情形下，在处理器中的一个或更多个为空闲（已将其量度设置为空闲值）的情况下，将使最接近PRN的父指针的处理资源实例来处理新任务，并且离该父指针最远的处理资源将出现长的空闲时长。

[0347] 当处理资源实例或者宏架构（即，通过时钟选通、电压 / 频率调制、功率隔离等）具有调节功耗的能力时，此行为是有用的。通过向软件客户机中介添加适当的驱动代码可以平衡这些特性。

[0348] 当必须重新唤醒处理资源时，不同的节能措施倾向于展现不同的影响，例如，时钟选通可能保持宏架构内的所有状态，而激烈的电压 / 频率缩放可能牺牲所有现存状态并且在被唤醒时出现不期望的涌入 (in-rush) 电流。在给定 PRI 具有多个不同代价的断电选项并且使用上述调度行为的情况下，根据空闲所占的时间来管理其使用是有意义的。

[0349] 根据处理资源实例和系统宏架构的能力，可将“空闲”状态分为多个子状态。从某些状态重新启动可能比从其他状态重新启动花费更大的代价（例如对比保持时钟选通断电的状态和功率隔离状态）。为了支持这些情形，SystemWeaver 支持多个空闲优先级。

[0350] 对于具有多个子状态的那些处理资源，在线程继续执行之前，中断响应会稳定地转变回空闲状态。这允许将处理资源逐渐地重新引入给定分配的活动组。

[0351] 示例

[0352] 图 29 示出了使用升序优先级随着时间进入空闲状态，处理资源实例 (PRI) 的“执行优先级”。在第一实例中，空闲任务将优先级设置为其可能的最低设置，给予该 PRI 当与其分配池同辈进行比较时可能从调度器分配任务的最高机会。

[0353] 一段时间后，空闲任务引发处理资源宏架构内可能支持的断电模式。此时，空闲任务提高了该 PRI 的执行优先级，以降低分配任务的可能性（在先前状态中的 PRI 将占先）。

[0354] 类似地，经过另一段时间之后，空闲任务（或某些其他代理）进一步提高了所述执行优先级，（可能对该 PRI 进行功率隔离，从而排除了静态泄漏）。调节所述优先级使该 PRI 对于任务分配的适任性更低（与重新唤醒该处理资源的代价（在此情况下为涌入流、冷高速缓存效应等）一致）。

[0355] 注意：对分配层中第一个条目的优先抢占有时是不期望的。在这种情况下，在前一调度判定成为后续调度操作的新起点的情况下，可以选择不同的行为。该选择展现了在分配池内的处理资源实例之间对高优先级抢占任务的更公平的分配。

[0356] 基于滞后的调度

[0357] 在某些情况下，期望保持调度判定，而不管其工作绝对正确性。通常，这是这样的情况：为特定类别的任务或数据集建立环境的代价高，因而应该在可允许的情况下在多个任务上聚合。这种情况的示例包括：

[0358] - 处理器高速缓存：针对历史算法或数据集而组装的高速缓冲存储器会显示出与全异算法 (disparate algorithm) 和 / 或数据集的较差亲和性 (affinity)。这被称为冷高速缓存效应并表现出高的高速缓存错失率 (miss ratio)，并因此表现出较差性能。

[0359] - 可重新配置的 FPGA 分区：部分运行时可重配性使 FPGA 的一部分能够在芯片已部署并工作的同时被动态地重新配置，使得能够随着时间而执行不同的算法。然而，从一个算法切换到另一算法的代价较高，从而必须在较大的数据集上聚合以确保系统效率。

[0360] 这两个都是代价高的环境切换的示例。

[0361] 基于滞后的调度可以通过跨多个用户操作聚合环境切换的代价从而用于避免某些负面效应。通过使用量度之一来表示“系统代价”参数，可以实现基于滞后的调度。该滞后量度可基于系统内的多个代价的测量：

[0362] - 任务存储器占用：在存储器非常重要的情况下，可以使用给定任务队列的累积足印来确定何时调度新配置。

[0363] - 处理要求：期望在实质上的“活动处理”时段上聚合环境切换代价的情况。

[0364] - 时间片：等待时间的抖动很重要的情况。

[0365] 例如，在被动态配置的 FPGA 的情况下，存储器可以为在可重新配置构造的单个部分上复用的各个算法环境累积工作；在此情况下，存储器占用可以是判定何时对阵列重新编程的因素。在所有情况下，可以将调度器层级设计为包容由于高优先级任务的到达而引起的被迫切换。

[0366] 以下总结了关键的系统级挑战：

[0367] - 环境切换的代价的影响（切换的时间、涌入流）

[0368] - 切换的定时

[0369] - 如何管理对给定环境（在其不活动的情况下）的工作的累积

[0370] 下面的章节描述使用 SystemWeaver 管理动态 FPGA 重新配置的可能的方法。使用类似的调度技术和更简单的软件客户机中介行为可以实现对冷高速缓存的管理。

[0371] FPGA 运行时间重新配置

[0372] 虽然与处理器环境切换非常类似，但仍需几个定义（重新定义）。

[0373] 配置：目标为 FPGA 构造的给定部分的一套编程变量。

[0374] 环境切换：改变 FPGA 的可重新配置的部分的配置的行动。

[0375] 任务：由给定 FPGA 配置执行的单个单位的工作。

[0376] 在该建议中，目标为 FPGA 的可重新配置的部分的配置被视为协作型（与抢占型相反），也就是说，个体任务是不可分的并且必须在环境切换能发生之前完成。这确保了任务不需要再进入并且将在环境之间的状态保持问题限制为这样一个问题，即，等待特定配置的任务的数量必须是整数值。图 30 示出了对未完结任务进行的管理的逻辑图。

[0377] 将任务组织进队列。这些队列永久存在；具体地，这些队列累积当前不活动的 FPGA 配置的工作。调度器确定何时切换任务并且管理任务组内任务的执行顺序。该重新配置支持逻辑管理对该构造进行重新编程的技巧，并在完成时发出信号。根据该模型的协作本质，当对环境切换进行调度时，没有需要保留在该构造内的数据。

[0378] 调度器

[0379] 调度器执行两个不同的功能：

[0380] - 调度器根据任务队列的变化状态持续地评估当前的调度判定。

[0381] - 调度器管理个体任务队列内任务的执行顺序。

[0382] 每个任务的到达都会引起调度判定的更新，以确保 FPGA 构造总是处于正确的状态（贪婪调度）。在任务队列内，调度器根据由系统设计师定义的属性制订执行顺序。调度器至少应该提供 FIFO、轮转和优先级策略。

[0383] 使用 SystemWeaver 进行重新配置管理

[0384] SystemWeaver 方案提供了一套丰富的调度能力和处理器间通信能力，可以部署所述能力来管理运行时间并行执行以及处理器间的通信。SystemWeaver 的特性可有效地管理传统指令集架构、固定的硬件元素和可重新配置的 FPGA 块等之内的任务和环境切换。图 31 示出了一示例架构：

[0385] SystemWeaver 处理对以下行为的管理：对 FPGA 构造的环境切换进行的调度；以及对个体任务队列内的任务进行的排序。自然地，除此之外，还有将更传统的任务调度到平台内的固定配置元素。

[0386] 用特定 SystemWeaver 硬件客户机中介处理重新配置自身。要注意，用于管理高速缓存的“温暖度 (warmth)”的类似调度技术对标准客户机中介没有另外的要求。将由客户机中介接收的各被调度任务控制块与该构造的现有配置进行比较。如果当前载入的配置与被调度任务的配置不同，则客户机中介重新配置该构造而不与 SystemWeaver 核进一步交互。然后，构造更新判定由调度器所规定的任务的输出顺序排他地进行控制。可以将客户机中介重新设计为包容不相似的重新配置策略。

[0387] 调度策略

[0388] 应该由系统设计者确定调度策略。然而，存在必须要有用来支持该特性的关键能力。具体地，调度算法应该可以表现出滞后，即，保持一种调度判断，直到别处已累积了用于保证切换到另一判定的足够代价为止。

[0389] 在示出的示例中，每个“任务”都随机地产生量度，该量度被添加到表示任务组的累积量度。当给定任务组由从队列中移除任务的“弹出”操作提供服务时，使该累积计数值递减。

[0390] 当“推送”（新任务到达）或者“弹出”操作发生时，该构造的调度器针对当前在执行的任务组的量度评估各个候选项。根据图 32 给出的算法：

[0391] 需要更新＝ ($C_{candidate} > C_{current}$+Hysteresis)

[0392] $C_{candidate}$ 候选调度层内的任务的累积代价。

[0393] $C_{current}$ 当前选定的调度层内的未完结任务的累积代价。

[0394] Hysteresis 为了避免环境颠簸 (thrashing) 而添加的滞后量。

[0395] 图 33 示出了可以选择用来实现上述算法的调度层级。在此情况下，“滞后调度器”的量度 1 存储滞后操作数。量度 0 可用于存储滞后组的静态优先级，当在滞后组和抢占组之间进行调度时使用该优先级。假设存在某些任务，这些任务有足够的优先级迫使环境改变。

[0396] 结果

[0397] 图 32 示出了这种系统的理论效果。任务调度输出在本质上是蓄意不均的 (blocky)，其最大程度的使用任何给定配置，同时管理对系统通信量成形的影响。

[0398] 图 34 示出了提出的调度算法的仿真结果。与所有代价的总和（累积的）一起画出了等待四个配置中的各个配置的任务的累积“代价”。选择的轨迹指示所述算法将选择哪一个可用配置。

[0399] 复合调度算法示例

[0400] 例如当对池分配层中的处理资源池进行调度时复合调度操作符是有用的。例如，成员的子集可能仅在等待处理的任务的队列饱和度超过一特定阈值时才会适任。

[0401] 考虑可使用三个处理资源、1 个 RISC 处理器和两个 DSP 的情况（图 35）。这些资源中的每一个在理论上都能够执行语音编码操作，但是 DSP 装置更有效率。在此情况下，如所示出的，RISC 处理器将存在于语音编码池中，但是其参与该执行的适任性将取决于等待该功能的任务的队列深度。

[0402] 在这种配置中，池根节点量度 0 可能表示优先级，而量度 1 可能表示队列饱和度。在各个候选 PSN 中，量度 0 通常将表示在其各自的处理资源示例 (PRI) 上运行的任务的执行优先级。在此情况下，量度 1 将表示使其相关联的 PRI 适任于调度所需的队列饱和度。在

此情况下，该复合调度算法为：

[0403] 需要更新＝ $(M_{current}0 > M_{candidate}0)\&\&(M_{current}1 > M_{candidate}1)$

[0404] 在与 DSP 装置相关联的 PSN 中，将 M1 设置为 0，从而该算法纯粹基于优先级来确定。在 RISC 处理器的情况下，M1 将是非零的，因此由 $M_{current}1$ 所表示的队列饱和度必须变得大于该值以使 RISC 处理器参与该算法的执行。

[0405] 下面在宏架构或事务级别上描述 SystemWeaver 服务器。

[0406] 如前所述，存在 SystemWeaver 硬件方案的四个元素：

[0407] -SystemWeaver 服务器核

[0408] - 与 SystemWeaver 紧密耦合的存储器

[0409] -SystemWeaver 调试管理器

[0410] -SystemWeaver 客户机中介

[0411] 全局量度代理是可选的，并且在系统设计需要在调度判定中包括系统全局状态时使用该全局量度代理。

[0412] 主要连接组

[0413] 图 38 示出了在核周围找到的接口组。

[0414] 为了确保可以容易地集成 SystemWeaver 核，所有信号都是单向的并且对于单个时钟是同步的。下面给出这些组的组成成员的细节。所有信号方向是相对于 SystemWeaver 核给出的。

[0415] 系统控制组

[0416] 系统控制组包含确保 SystemWeaver 核的正确操作所需的各种各样的信号。这些信号包括系统时钟、实时时钟以及重置信号。

[0417] 全局量度组

[0418] 在某些系统中，期望在调度判定期间使用特定的系统量度。这些量度可表示各种因素，诸如互联繁忙、高速缓存命中率、存储器占用等。

[0419] 外围中断组

[0420] 外围中断组包括源自 SystemWeaver 控制的系统的外部的一组中断。例如可以从与外部世界的输入接口或者直接经由针脚从 SoC 装置的外部驱动外围中断组中的信号。在 SoC 设计时定义外围中断输入端的数量。

[0421] 内部中断组

[0422] 内部组包括由 SystemWeaver 系统发起的两组同步中断以及一组运行时间系统调试信号。信号组内各个信号的数量通常会对应于系统内处理资源的数量，并且在 SoC 设计时定义。

[0423] 紧密耦合存储器接口组

[0424] 该组将 SystemWeaver 连接到其自身私有的紧密耦合存储器资源。假设附加的存储器是同步 SRAM 装置。在 SoC 设计时定义地址路径的宽度 n 和数据路径的宽度 m。

[0425] 互联组

[0426] 必须在 SoC 设计时设置单独的互联策略，其包括协议以及层的数量。可以在对应的总线特定的实现中找到任意给定总线接口信号的细节。

[0427] 调试接口组

[0428] 对于到调试管理器的接口的细节，请参见共同待定的国际 PCT 申请第 PCT/GB2005/003525 号，在此通过引用将其并入。

[0429] 紧密耦合存储器 (TCM)

[0430] SystemWeaver TCM 是由多个 EDA 制造商提供的一种标准编译器 SSRAM 技术。TCM 包含在 SoC 设计时根据应用的需要而定义的整数数量的 SystemWeaver 存储器元件 (WME)。每个 WME 消耗 256 位的存储器空间。SystemWeaver 支持最大 65536 个 WME 或 16Mb 的存储器。

[0431] 虽然队列描述符不消耗 WME，但是在通常的系统中，所需 WME 的数量将由线程支持要求所支配。例如，能够支持同时位于 SystemWeaver 服务器内的 400 个线程的系统将需要大约 128kb 的附加存储器。

[0432] 在 SoC 设计时，可以修改存储器接口以简化路由。

[0433] 服务器核的子块的描述

[0434] 图 39 示出了 SystemWeaver 服务器实体的主要逻辑组件。这些功能映射到图 40 所示的架构上。在执行以下功能的四个主要内部并行处理元素之间分配功能：

[0435] - 线程调度器输入管理器 (TSIM)：空闲链表维护、WME 恢复。

[0436] - 线程调度器待定管理器 (TSPM)：待定链表维护、同步、推进至就绪队列结构。线程同步管理器保持待定队列结构的完整性（插入和提取）。

[0437] - 线程调度器输出管理器 (TSOM)：就绪队列维护、派出队列维护、处理资源功率管理、中断产生。保持就绪队列结构的完整性（插入和提取）。

[0438] - 线程调度器调度管理器 (TSSM)：维护就绪队列结构内各个处理资源的调度判定。

[0439] 此外，一些块提供支持功能：

[0440] - 线程调度器存储器管理器 (TSMM)：聚合对附加的 SystemWeaver 存储器的访问，包括互斥和加锁。

[0441] - 线程调度器中断管理器 (TSIC)：将进入系统的中断转换为内部同步原语。

[0442] - 线程调度器接口管理器 (TSIF)：向 SystemWeaver 资源提供互联接口和配置以及运行时间访问。

[0443] 图 40 示出了 SystemWeaver 架构的主要子块。下面的章节详细描述了在这些组件之间进行的子块内交互。每个子块向其他子块提供一组“公用方法”，使每个子块能够指示其同等物对它们各自保持的结构执行操作。

[0444] 在一命令会以特定条件完成的情况下，在子块内管理状态标志。

[0445] 子块接口图的箭头方向指示总线的主要运量 (master-ship)，而没有反映信号组的个体元素的方向。

[0446] 全状态 (stateful) 描述符行为

[0447] 在 SystemWeaver 的操作期间使用了多种描述符类型（对于进一步的细节，请参见共同待定的国际 PCT 申请第 PCT/GB2005/001154 号，在此通过引用将其并入）。这些描述符中的大多数是无状态的，然而，在特定情况下，线程描述符和调度器描述符会在多种状态中转变。此文档描述了这些状态转变以及引起这些状态转变的事件。

[0448] 线程描述符

[0449] 存在由 SystemWeaver 内部识别的两种类型的线程描述符：标准线程描述符和标记线程描述符。后者专门用来：对移除调度层级的处理进行同步，同时保证先前排队的线程描述符的完整性和顺序。

[0450] 图 41 示出了内部线程的状态图，线程描述符和标记线程描述符都遍历该状态图。要注意，状态“新建”和“释放”是元状态，它们与 SystemWeaver 内的永久状态没有直接相关性。在线程描述符内不存在原意状态变量，代之地，状态由多个标志表示。表 3 给出了标志状态与其中存在图 41 中的线程的状态之间的相关性。

[0451] 标准线程描述符的状态描述

[0452] 下面的章节给出对状态以及引起进入该状态以及从该状态离开的事件的简要描述。

[0453] 新建

[0454] 新建状态是瞬时的。根据独立于或依赖于线程的命令的推送将新线程引入 TSIF。如下处理这两种情况：

[0455] 一独立线程（没有时间或同步依赖性的线程）立即转变到推送状态。在该实例中，TSIF 指示 TSSM 将该线程引入就绪队列结构。

[0456] - 依赖线程（具有时间或同步依赖性的线程）转变到阻塞状态。TSIF 指示 TSPM 适当地将该线程引入待定和定时队列结构。

[0457] 阻塞

[0458] 在阻塞状态下，线程描述符等待外部同步和 / 或基于定时的同步。阻塞线程在 TSIF 发起。当接收到适当的同步时，TSPM 将该线程转变到推送状态并指示 TSSM 将该线程引入就绪队列结构。

[0459] 推送

[0460] 推送状态中的线程要么已被同步，要么原本就是独立的，TSPM（依赖线程）和 TSIF（独立线程）将在各个情况下管理至推送状态的转变。TSSM 将该线程推送至就绪队列结构并将该线程转变到就绪状态。转变到就绪状态引起重新调度的发生。

[0461] 就绪

[0462] 就绪状态中的线程要么是从推送状态转变的，要么是被冲出 (flush) 回到就绪状态结构的（由 TSSM）。转变到就绪状态总会引发重新调度。线程可从就绪状态转变到弹出状态或冲出状态，后者是这样的特定情况的结果，即，经过弹出情况以在单次操作中冲出。当线程被 TSSM 指派为用于调度的最适任候选时，线程转变到弹出状态。

[0463] 弹出

[0464] 弹出状态中的线程已被调度器 (TSSM) 指派为对于由一特定处理资源实例或一组实例进行的处理最适任的线程，这些线程是被 TSOM 转变到该状态的。线程可以从弹出状态转变到冲出状态或僵尸状态 (zombie state)：

[0465] - 由于进行重新调度而识别出更适任的线程，结果 TSOM 将线程转变到冲出状态。

[0466] - 由于开始在系统处理资源实例之一中进行处理，TSOM 将线程转变到僵尸状态。僵尸状态保持线程描述符，直到其可被释放为止。

[0467] 僵尸 (zombie)

[0468] 在 TSSM 内处理僵尸线程。僵尸状态的存在是为了保证在释放线程之前已经免除

了针对给定线程描述符的所有依赖性。一旦线程到达 TSSM 处理队列的前部就可保证此点，因此不需要进一步的处理。

[0469] 冲出

[0470] 由 TSSM 处理被冲出的线程。必须将冲出的线程重新引入就绪队列结构，从而引起重新调度操作。一旦完成了重新引入，TSSM 就会将该线程转变回就绪状态。

[0471] 释放

[0472] 释放状态是一瞬时状态，其指示该线程描述符所消耗的 WME 被放回了空闲链表。

[0473] 标记线程描述符的状态描述

[0474] 下面的章节给出对状态以及引起进入该状态以及从该状态离开的事件的简要描述。

[0475] 新建

[0476] 新建状态是瞬时的。根据推送标记线程命令将新标记线程引入 TSIF。标记线程总是经过阻塞状态，以确保在标记线程自身到达之前存在于 TSPM 的输入和输出工作队列中的任何处理都已完成，所述处理会影响标记线程最终删除的调度器层的状态。

[0477] 阻塞

[0478] TSPM 直接将标记线程转变到推送状态并指示 TSSM 将该线程引入就绪队列结构。

[0479] 推送

[0480] TSSM 将标记线程推送至就绪队列结构并将该线程转变到就绪状态。转变到就绪状态引起重新调度的发生。

[0481] 就绪

[0482] 当标记线程被引入就绪队列结构时，其对其直接父亲解锁。然后在依赖线程的数量的计数值的控制之下解放该父调度层。标记线程仅在其父亲的依赖线程计数值到达零时（即 SystemWeaver 内没有依赖于该父调度层的存在的其他线程描述符）才适任于离开就绪状态的转变。

[0483] 与标准线程相同，标记线程可从就绪状态转变到弹出状态或冲出状态，后者是这样的特定情况的结果，即，经过弹出条件以在单次操作中冲出。当标记线程被 TSSM 指派为用于调度的最适任候选时，标记线程转变到弹出状态。

[0484] 弹出

[0485] 弹出状态中的标记线程已被调度器 (TSSM) 指派为对于由一特定处理资源实例或一组实例进行的处理最适任的线程，要注意，该调度判定是一特殊情况，其指的是调度层为空并且不存在其他依赖线程。标记线程可以从弹出状态转变到冲出状态或僵尸状态：

[0486] - 由于进行重新调度而识别出更适任的线程，结果标记线程被转变到冲出状态。

[0487] - 由于开始在系统处理资源实例之一中进行处理，标记线程被转变到僵尸状态。僵尸状态保持线程描述符，直到其可被释放为止。此外，标记线程的父亲也被标记以在该状态期间删除。

[0488] 僵尸

[0489] 对僵尸标记线程的处理与正常线程描述符的类似。在该状态内还删除标记线程的父调度层。

[0490] 释放

[0491] 释放状态是一瞬时状态，其指示该线程描述符所消耗的 WME 被放回了空闲链表。

[0492] 调度器层状态图

[0493] 调度器层还具有暗含状态。对于静态层，即，在系统的整个运行时间持续存在的层，唯一的状态是活动状态。其余的状态由动态调度层使用，即，在运行时间期间来来往往的层。表 4 给出了标志状态和图 42 中的其中存在调度器层的状态之间的相关性。

[0494] 调度器层描述符的状态描述

[0495] 下面的章节给出对状态以及引起进入该状态以及从该状态离开的事件的简要描述。

[0496] 新建

[0497] 新建状态是瞬时的。在初始化期间或在运行时通过一推送独立元素将新调度器层引入 TSIF。

[0498] 休眠

[0499] 在休眠状态中，允许调度器层累积线程，并且潜在地，允许调度器层累积另外的子层级，但是从来不会被 TSSM 调度。可以以两种方式进入该休眠状态：

[0500] - 在休眠状态中可以创建新调度描述符。

[0501] - 调度层可在运行时修改并根据 TSIF 发出的明确的系统命令而置入休眠状态。

[0502] 仅可通过明确的系统命令离开休眠状态。

[0503] 活动

[0504] 在活动状态中的调度器层活跃地参与调度并被加锁，这意味着仅当其变空时才将其移除。通常在该状态中创建静态调度器。根据通过 TSIF 接收到的明确的系统命令，动态调度器转变到该状态。仅当接收到标记线程时调度器层才离开活动状态，调度器层进入“待释放”状态。

[0505] 待释放

[0506] 调度器层停留在待释放状态中，直到以下两个标准都满足为止：

[0507] - 依赖元素的数量（其是保持测量的对引用该元素的描述符的数量的测量结果的计数值）变为 0

[0508] - 子元素的数量变为零

[0509] 要注意，这意味着引起转变到待释放状态的标记线程描述符也被弹出，仅在调度器层没有其他依赖性时才会发生的行为。

[0510] 释放

[0511] 释放状态是一瞬时状态，其指示调度层描述符所消耗的 WME 被放回了空闲链表。

[0512] 子块间的行为

[0513] 下面的章节覆盖了涉及多个子块的 SystemWeaver 行为的多个方面。

[0514] 动态调度层级的操作

[0515] 在系统运行时间期间可添加和移除调度层级。如果涉及到特定用户级过程，则 SystemWeaver 保证系统的完整性以及从调度层级的一部分转移到另一部分的线程描述符的顺序。对于进一步的细节，请见上文。

[0516] 动态调度量度的更新

[0517] 可以更新标准线程或标记线程内的量度。行为依赖于线程状态：

[0518] - 如果线程处于阻塞状态，则将适当的命令发送到 TSPM。因为待定队列是经排序的，所以当更新了量度时，从该队列中移除该线程或者将该线程重新插入到该队列中，以确保该线程在适当的点重新出现。

[0519] - 如果线程处于任何其他持久状态，则向 TSSM 发出执行量度更新并且对调度层级的适当部分进行重新调度的命令。

[0520] 用于调度的 TSOM/TSSM 交互

[0521] 在线程描述符到处理资源实例的调度中，TSOM 和 TSSM 都起了作用。在图 72 和 73 中给出了 TSOM 和 TSSM 之间的交互的一些示例序列图。

[0522] 子块内架构和行为

[0523] 就其主要 IO 或到其同等物或外部世界的物理接口，并且还就其命令接口或使用适当物理接口上的命令协议而可以调用的方法，来讨论各个子块。

[0524] TSIF- 接口管理器

[0525] 接口管理器负责协调 (orchestrate) 从互联组接收的命令的执行，并将其分配给其他子块。下面的章节描述存在于 TSIF 内并且超越对内部消耗的命令简单翻译的功能实体。

[0526] 架构

[0527] TSIF 主要将通过互联接口接收的命令解释 (interpret) 成用于其余子块的一个或多个内部命令。下面的章节给出了存在于 TSIF 内的架构资源的细节。

[0528] 信号机区域锁

[0529] 信号机区域锁 (SRL) 提供了这样的资源，其可由任何系统资源原子地测试和加锁以获得对系统资源的排他访问。可能出于以下任何数目的原因而使用 SRL：

[0530] - 为了锁住系统存储器中包含 - 个或更多个共享资源（例如，信号机对象、任务控制对象等）的区域，从而保证了完整性。

[0531] - 为了锁住用于多循环命令访问的 SystemWeaver 命令接口

[0532] - 为了锁住用于多循环事件的 SystemWeaver 调试事件接口

[0533] SRL 有两种状态：已加锁和已解锁。读取 SRL 被定义为尝试获得锁，对 SRL 写被定义为尝试解锁。在锁住特定 SRL 的处理资源实例和释放该特定 SRL 的处理资源实例之间没有必需的相关性。下面描述了该行为：

[0534] - 已解锁。在已解锁状态中，对 SRL 的读取返回向读取者指示该加锁尝试是否成功的控制码。在该状态中，写没有效果。

[0535] - 已加锁。在已加锁状态中，对 SRL 的读取指示该 SRL 不可用。写会释放该 SRL。

[0536] 命令处理

[0537] TSIF 将从系统互联接收的命令分成可能多个内部命令。

[0538] Watchdog 和时间片支持

[0539] 可选地，可向每个处理资源实例提供双模式定时器计数器。这两个定时器模式是 watchdog 和时间片，缺省模式为 watchdog。

[0540] 由 watchdog 循环和个体计数值定义 watchdog 行为（图 44）。Watchdog 是系统时钟的逐步减低 (step down) 版本，由硅设计时的系统常量来定义该逐步递减。图 46(a) 示出了每个预置 (prescaled) 的 watchdog 时钟周期内每个处理资源实例的行为：

[0541] - 询问个体 watchdog 计数器以确定 watchdog 中断是否恰当。

[0542] - 如果有必要，则产生 watchdog 中断。

[0543] 来自处理资源实例（定时器与之相关的）的每个控制访问将重置 watchdog 定时器计数值，因此，只有在 watchdog 时间间隔不访问 SystemWeaver 的处理资源才经历 watchdog 中断。

[0544] 在 TSIF 和 TSOM 之间划分前述时间分片行为。在 TSIF 中，分配给各个处理资源的定时器可以是专门用于时间片支持的。当出现时间片间隔时，在 TSOM 中提供自动量度操作（运行优先级被递减）。仅当由于自动的量度更新从而标准抢占变为适当时，处理资源自身才被抢占。

[0545] 图 45 示出了 TSIF 内时间片支持逻辑的每系统时钟周期的行为。图 46(b) 示出了每个预置的时间片时钟周期内每个处理资源的行为。要注意，在发生了时间片事件后，定时器模式回到 watchdog 模式。

[0546] 中断处理

[0547] SystemWeaver 接受来自系统的中断、外围中断、并向系统发起中断（处理资源实例中断）。

[0548] 外围中断

[0549] 在 TSIF 内可掩蔽外围中断以及对外围中断设置条件（边沿 / 电平触发、负 / 正逻辑等）。

[0550] 处理资源实例中断

[0551] 在 TSIF 内设置中断处理资源以提供以下便利：

[0552] - 保持中断状态，包括中断断言的源（抢占、时间片、watchdog)。

[0553] - 掩蔽能力。

[0554] 在 TSIF 内实现了对 SystemWeaver 命令接口自动进行中断掩蔽的特殊特性。因为命令接口是系统内所有处理资源的公共访问点，所以必须保持完整性和效率。出于此目的，当一处理资源获得了对 SystemWeaver 命令接口的锁定时，SystemWeaver 自动处理中断掩蔽以确保该处理资源不会在代码的该关键区期间被中断。

[0555] 针对跟踪已接收到的成功 SystemWeaver 命令接口信号机区加锁请求的数量的各个处理资源实例，保持一计数器。每个加锁请求使计数值递增，而每个解锁请求使其递减。当计数器从零递增时，中断被自动掩蔽，当计数器递减到零时，中断被自动解掩蔽。

[0556] TSIM- 输入管理器

[0557] 输入管理器管理 WME 空闲链表，处理来自 TSIF 的弹出请求以及来自多个子块 (TSIF、TSPM、TSOM、TSSM) 的推送请求。

[0558] 架构

[0559] TSIM 仅包含一个架构实体，即，SystemWeaver 存储器元件 (WME) 空闲链表。图 48 示出了该链表的结构。

[0560] 空闲链表根据后进先出 (LIFO) 策略进行工作。该链表的每个成员都是 C_SCHED_ENTRY_FREE 类型的，并且使用由 C_FREE_QUEUE_POINTER_INDEX 所引用的指针单向链接。

[0561] 方法接口

[0562] 除了基本的链表维护方法（取得和设置状态）之外，输入管理器还提供关于其接

口的以下命令：

[0563] Push Free Index(C_TSIM_CMD_PUSH_INDEX)

[0564] 调用者：(TSIF、TSSM、TSOM、TSPM)

[0565] push free index 命令用于将所释放的 WME 索引推送回空闲链表。下面总结了参数：

[0566]

unsigned short Index	新释放的 SystemWeaver 存储器元件的索引。

[0567] Pop Free Index(C_TSIM_CMD_POP_INDEX)

[0568] 调用者：TSIF

[0569] pop free index 命令用于从空闲链表弹出一空闲 WME 索引。下面总结了参数：

[0570]

unsigned short*pIndex	对设置新分配的WME节点ID的位置的引用。

[0571] TSPM- 待定管理器

[0572] 待定管理器管理等待某些事件而被阻塞的任务或线程描述符；其要么是基于同步的，要么是基于定时器的。个体待定队列关联由用户控制（或者由更高层软件控制），其可表示信号机、竞争域、中断或它们的任何组合。

[0573] 架构

[0574] 待定管理器包括两个主要元素：待定队列和定时器队列的可变数量。待定队列存储等待由某些外部事件进行同步的线程的链表，而定时器队列存储等待超时的线程的链表。线程描述符可能（实际上普遍地）是这两个链表的成员，由此允许线程描述符等待外部同步事件一段有限长度的时间。

[0575] 待定队列结构

[0576] 图 50 的待定队列结构主要出现在紧密耦合存储器内，用 TSPM 内的资源和能力来处理其内容。TSPM 自身包含一头指针和引用待定队列描述符的链表的多个元素（即，链表的链表）。每个待定队列包含线程描述符的链表，待定队列的数量在运行时可动态地增长和减小。所有阻塞状态中的线程描述符存在于待定队列中（与定时器队列相反，在定时器队列中仅当线程具有定义的超时时该线程才出现）。对多个待定队列的使用依赖于应用和应用程序员的需要和偏好。可将待定队列与以下事务等相关联：

[0577] - 信号机。这可能导致各自包含很少线程的大量待定队列。因此，在这些情形下，最差情况同步响应时间会很低。

[0578] - 竞争域。竞争域是竞争相同资源的多个实体内的区域。例如，可将进程（相对于线程）视为竞争域。

[0579] - 中断。为了得到最快响应事件，通常将中断聚合在一起成为排他专用的待定队列。

[0580] 与定时器队列不同，待定队列是纯粹基于事件的。这些事件是：

[0581] - 推送事件，其中，将新线程引入一待定队列，该待定队列要么已经存在，要么必须被创建。

[0582] - 同步事件，其中，一个或更多个线程必须转移进就绪队列结构。

[0583] 下面的章节描述在这些情况下的行为：

[0584] 推送事件

[0585] 对于推送事件，TSPM 必须（根据所识别的排序操作符 metric[0]）确定在链表的何处插入线程描述符。必须考虑两种情况：

[0586] - 推送至现存的待定队列

[0587] - 推送至新的待定队列

[0588] 前一情况是普通的，按顺序询问链表，直到找到插入点为止。在升序优先级的经典情况下，将排序操作符设置为“大于”(C_PEND_MNGR_PUSH_GTR) 并搜索现存的链表。由新线程的 metric[0] 大于下一链表成员的 metric[0] 时的点来定义插入点。

[0589] 当需要新的待定队列时，就在插入新线程条目之前，发生待定队列的插入。

[0590] 同步事件

[0591] 可以从中断控制器 (TSIC) 或通过命令接口接收同步事件。同步可以以两种模式发生：

[0592] - 原意模式，其中，命令参数对 WME 索引进行原意引用。

[0593] - 相关模式，其中，在线程描述符内的字段和同步原语内的字段之间寻求相关性。

[0594] 在原意模式下，因为将线程描述符的索引传递给命令，所以不需要搜索最适任候选项。相关模式需要首先找到最适任同步候选项。

[0595] 相关模式包括三个子类型：

[0596] - 正常，其中，只有指定待定队列内的最适任候选被同步。

[0597] - 多播，其中，指定待定队列内的所有适任候选被同步。

[0598] - 广播，其中，所有待定队列内的所有适任候选被同步。

[0599] 通过与命令传递的参数以及也在命令内的元素标识符来识别最适任候选。该元素标识符规定用哪一个候选线程描述符的字段与所传递的参数进行比较以识别适任性。在正常模式下，算法向下对待定队列叠代，直到找到一适任候选项为止，在该点处将该适任候选项从该等待队列移除（在适用的情况下从定时器队列移除），并转发至就绪队列结构。对于多播和广播模式，该处理继续直到该等待队列或各个等待队列分别穷尽为止。

[0600] 特殊条件与移除定时器队列的成员相关联。

[0601] 参见下面的章节以得到更多细节。

[0602] 定时队列的结构和操作

[0603] 被引入超时队列的每个新线程最初包含一绝对超时值。该值可以是接口管理器从作为参数接收到的 32 位相对或绝对超时中得出的。

[0604] 定时器队列使用 C_THREAD_PENDING_QUEUE_TIMER_PTR_INDEX 来存储按其超时排序的线程描述符的链表，最接近的超时位于链表的头部。要注意，通过由 C_THREAD_PENDING_QUEUE_PRIORITY_PTR_INDEX 所索引的第二组指针，这些线程描述符也会是优先级列表的成员。图 51 示出了超时队列的基本结构。在定时器队列的个体线程描述符成员内并不存储绝对超时值，而是存储与直接前任的超时的相对超时。使用浮点表示将该值存储在一 16 位字段内，其中，浮点定时器格式的尾数包括带有 5 位指数的 11 个有效位的绝对数。总是将定时器队列的头元素的超时字段复制到 TSPM 的 TimerHeadTimeout 寄存器，并在之

后重置该超时字段。

[0605] 定时同步架构的操作使用几个持久的内部资源：

[0606] -TimerHeadIndex；定时器链表的头指针

[0607] -TimerNumElements；定时器队列中的元素的数量

[0608] -TimerHeadTimeout；头元素的超时的快照

[0609] -TimerDivider；系统时钟的预置器

[0610] -TimerDividercounter；分割器（divider）的倒计时资源

[0611] -TimerCounter；根据每个预置的时钟滴答（tick）而单调增加的32位计数器资源

[0612] -TimerErrorAdjustCounter；用于累积和包容错误的32位计数器

[0613] 如果TimerDivider寄存器被设置为0，则禁用了定时器功能。

[0614] 定时器队列循环行为

[0615] 图52示出了在各预置的时钟滴答发生的操作。当没有基于待定定时器的同步（定时器队列处于等待状态）时，TimerHeadTimeout为非零，因此不采取行动。当TimerHeadTimeout变为0并且定时器队列不为空时，系统根据TimerErrorAdjustCounter的值采用两个状态中的一个状态。如果TimerErrorAdjustCounter为零，那么在此循环发生TimerHeadTimeout的期满，并且创建一定时器同步原语，该原语将最终引起从该定时器队列（和用于维护目的的优先级队列）的弹出。之后，TimerErrorAdjustCounter立即单调递增，直到完成了对时间事件原语的处理之后将其重置为止。

[0616] 定时器队列事件

[0617] 存在引起定时器队列操作的三种事件：

[0618] -时间事件原语（C_TSPM_CMD_TIME_PRIMITIVE）

[0619] -超时为非零的线程推送事件（超时设置为零的线程未放置在定时器队列中）

[0620] -非基于定时器的同步事件，其导致从定时器队列中移除线程（作为维护行为）。

[0621] 图53示出了超时逻辑的操作模式的非常基本的表示。当在等待状态中时，TimerHeadTimeout为非零并且根据预置的时钟单调递减。在该状态中，TimerErrorAdjustCounter保持为零。当TimerHeadTimeout到达0时，定时器队列的头已超时，并且FSM转变到活动状态，在活动状态中为弹出操作提供服务。在该状态内，TimerHeadTimeout为零并且TimerErrorAdjustCounter在每个循环单调增加。该错误计数值用于确定对前一超时事件采取行动所花费的时间是否使得后续超时事件适任。一旦没有未完结的其他适任超时事件，则FSM转变回等待状态并且TimerErrorAdjustCounter被重置。

[0622] 从定时器的潜在序列得到的弹出操作中的第一个是根据线程描述符内的重置超时字段推断出的（见图52）。作为定时器队列的头，该线程总是被弹出。之后，TSPM必须评估随后的弹出是否适任，为了使其容易，使用另外的资源TimerLastError来保持所有累积弹出线程描述符的超时增量（delta）之和。对于定时器队列的后续成员的各个进一步的叠代，从TimerErrorAdjustCounter中减去TimerLastError，创建归一的错误计数值，将其与位于定时器队列头部的新线程描述符的超时进行比较。如果该线程内的超时增量小于所述归一的错误计数值，则也应该弹出该线程描述符。TimerLastError最初为零，因此，将线程超时增量直接与TimerErrorAdjustCounter进行比较。图54示出了在线程1的超时已流

逝并且相关的弹出操作已发生之后的前一定时器队列结构。要注意，TimerLastError 已被线程 2 的增量更新，线程 1 弹出操作的持续时间意味着线程 2 现在也适任了。

[0623] 图 55 示出了在弹出线程 2 之后队列的状态。要注意，已将线程 2 的增量添加到 TimerLastError，创建了线程描述符增量之和的运行中的累积。还要注意，对线程 2 的弹出操作花费了足够长的时间，从而线程 3 现在适任了。

[0624] 图 56 示出了在弹出线程 3 之后的状态。在该情况下，后一线程（线程 4）不适任于弹出，所以定时器队列的状态可返回至等待。必须如所示出地重置 TimerHeadTimeout。

[0625] 要注意，当从活动状态转变回等待状态时，必须正确地重置 TimerHeadTimeout。这是通过从新定时队列头的增量中减去 TimerErrorAdjustCounter 和 TimerLastError 之间的差而实现的。

[0626] 当新线程被引入并推送至定时队列时，必须考虑两种情况：推送至定时队列的头部以及推送至定时队列体内。

[0627] - 对于推送至头部，将 TimerHeadTimeout 简单地设置为线程增量。在队列为非空的情况下，将旧的头描述符增量设置为 TimerHeadTimeout 减去新线程增量。

[0628] - 对于推送至体内，定时器队列块必须巡视定时器链表以识别插入线程描述符的位置。然后调节链表中的下一个的增量以包容增加新线程的增量（从下一线程增量中减去该新线程增量）。

[0629] 弹出操作

[0630] 无论是由于定时器还是由于外部事件同步事件而引起的弹出操作都类似地进行处理。要考虑两种情况：在定时器队列的头部弹出线程的情况、以及不是这种情况的情况。在前一情况下，存在三种情形：定时器队列处于等待状态、以及定时器队列处于活动状态。

[0631] - 对于从“等待中”定时器队列的头部进行的弹出操作，将 TimerHeadTimeout 添加到定时器队列的下一成员的定时器增量以形成新的 TimerHeadTimeout（要注意，在基于定时器的弹出中，TimerHeadTimeout 的值将总是为零）。

[0632] - 对于从“活动”定时器队列的头部进行的并且 TimerErrorAdjustCounter 大于下一线程描述符中的增量（即，下一线程适任于基于定时器的同步）的弹出操作，使错误计数器 (TimerErrorAdjustCounter) 重新基于所弹出的线程的增量。

[0633] - 对于从“活动”定时器队列的头部进行的并且 TimerErrorAdjustCounter 不大于下一线程描述符中的增量（即，下一线程不适任于基于定时器的同步）的弹出操作，错误计数器使将该增量递减，并且用该结果来更新 TimerHeadTimeout。定时器队列有效地回到等待状态。

[0634] 在所弹出的线程描述符不是位于定时器链表头部的情况下，必须将该定时器队列中下一线程的增量递增当前被移除的线程内的增量那么多。

[0635] 方法接口

[0636] 除了用于待定队列和定时器队列的基本链表状态操作（取得和设置状态）之外，待定管理器还提供了针对其接口的以下命令：

[0637] Synchronisation primitive(C_TSPM_CMD_SYNC_PRIMITIVE)

[0638] 调用者：TSIF，TSIC

[0639] synchronisation primitive 命令发出控制包，该控制包释放存储在待定队列内

的阻塞的线程描述符。下面示出了参数：

[0640]

unsigned short PendingQueueID	同步原语与之相关的待定队列的标识。
unsigned char Type	同步的类型。以下任一： - 单播 – 对来自所引用的待定队列的第一个匹配线程进行同步。 - 多播 – 对来自所引用的待定队列的所有匹配线程进行同步。 - 广播– 对来自所有待定队列的所有匹配线程进行同步。
unsigned long	引用。 同步原语与之相关的依赖引用。

[0641] Add thread to pending queue(将线程添加到待定队列)

[0642] 调用者:TSIF

[0643] 该命令将线程描述符添加到新的或现存的工作队列。该命令向待定管理器暗示：工作队列中存在线程描述符。下面的表例示了线程描述符中与该命令相关的字段。

[0644]

CREATE_NEW_PQ_FLAG	确定是否必须为该线程创建新的待定队列。
unsigned short PendingQueueID	向其添加线程描述符的待定队列。
unsigned long metrics	线程的在对线程描述符的插入进行排序时使用的适任性。
TierID	当该线程描述符被解锁时将其入队列到的就绪队列调度层。 用于设置调度层内的依赖线程计数
DependancyTimeout	用于在定时器队列内排序的超时。

[0645] Process marker thread(处理标记线程)

[0646] 调用者:TSIF

[0647] 该命令简单地将标记线程传递到调度管理器工作队列。使用其以确保在标记线程引起调度器拆卸(tear-down)的任何可能之前所有依赖线程都已被待定管理器处理。

[0648] Synchronisation primitive(同步原语)

[0649] 调用者:TSIF

[0650] 该命令向指定待定队列发出同步原语。下面的参数出现在命令结构中：

[0651]

WME Index	可使用该参数通过对其 WME 地址的原意引用来释放 WME。
Pending queue index	要应用的待定队列
Synchronisation reference	要进行比较以阻塞指定待定队列的线程内的引用的同步引用。
Synchronisation type	同步的类型。以下任一： - 单播 – 对来自所引用的待定队列的第一个匹配线程进行同步。 - 多播 – 对来自所引用的待定队列的所有匹配线程进行同步。 - 广播– 对所有匹配线程进行同步。

[0652] Update metrics(更新量度)

[0653] 调用者 :TSIF

[0654] 该命令更新被阻塞线程的量度并使适当的待定队列重新排序。如果所识别的线程不再阻塞,则可将该命令传递给 TSSM。下面的参数出现在命令结构中 :

[0655]

Metrics	所识别的线程的新量度

[0656] Unlock pending queue(解锁待定队列)

[0657] 调用者 :TSIF

[0658] 该命令将待定队列解锁,从而当该待定队列变为空时,可将其释放回空闲链表。下面的参数出现在命令结构中 :

[0659]

WMEIndex	要解锁的 WME 索引

[0660] TSOM- 输出管理器

[0661] 输出管理器管理派出队列结构(其引用下一执行中线程的描述符)和当前在执行的线程的执行量度。

[0662] 1.1.1 架构

[0663] TSOM 的架构以图 58 所示的派出队列结构为中心。输出管理器保持派出队列描述符的链表,其中,每个 DQD 都通过 ProcElementID 字段与系统处理资源实例相关联。DQD 存在两组元素,其紧密引用 TSOM 整体的功能。包括 ProcElementID、量度 0、量度 1 以及缺省量度 0 的第一组(以执行为中心的元素)引用任务的执行状态并且在任务的执行状态内进行管理。包括根调度器索引、抢占索引以及下一抢占索引的第二组(以就绪队列为中心的元素)引用等待执行的线程的就绪队列结构。TSOM 还管理向系统内的处理资源实例发送带外信号(通常为中断)以及从就绪队列结构弹出线程描述符。

[0664] 以执行为中心的元素

[0665] 下面是对派出队列描述符内以执行为中心的元素的用途的简要描述 :

[0666] -ProcElementID：其是存储索引的静态字段，所述索引引用派出队列描述符与之耦合的处理资源实例。

[0667] - 量度 0、1 是动态更新的字段，用于存储当前在执行的任务（包括“空闲任务”和潜在的多个断电状态）的执行量度。

[0668] - 缺省量度 0 是静态字段，用于支持优化，由此，当将当前在执行的线程推送进 SyetemWeaver 服务器并因此根据定义变为空闲时，可自动恢复空闲量度。

[0669] 以就绪队列为中心的元素

[0670] 下面是对派出队列描述符内以就绪队列为中心的元素的用途的简要描述：

[0671] - 根调度器索引是对与派出队列相关联的调度根层的静态引用。

[0672] - 抢占索引是动态字段，其存储给定处理资源的下一次执行的当前最适任候选项。抢占索引完全在 TSOM 内进行管理，并且在适当的情况下被设置为派出队列事件的结果。

[0673] - 下一抢占索引是动态字段，其存储调度层级内下一最适任线程的父亲或者该线程索引自身。下一抢占索引仅由 TSSM 设置并且用作向 TSOM 通知最适任线程在就绪队列结构内的位置的工具。TSOM 通常在处理后重置该字段。

[0674] 派出队列事件

[0675] 派出队列事件因两种原因发生：

[0676] - 重新调度事件：只要当调度管理器 (TSSM) 内的重新调度操作识别出通常是对就绪队列事件进行处理（推送、弹出或量度操作）所引起的派出队列所需的状态变化（即抢占）时，就会发生派出队列事件。

[0677] - 派出队列弹出事件：由抢占索引所引用的线程索引已被接口管理器 (TSIF) 弹出。利用派出队列描述符自身内的“派出队列弹出”标志（由 C_DISPATCH_DESC_POPPED_FLAG 所索引）向 TSOM 发信号以通知该事件。

[0678] 在派出队列弹出事件的情况下，被弹出线程转变到僵尸状态（见图 41），并且如果其不是已经存在于那里，则将其推送回工作队列，以由 TSSM 释放。之后，TSOM 解除该线程与就绪队列层级的链接。

[0679] 对于重新调度事件和派出队列弹出事件两者，对派出队列事件的处理继续下去以发起派出队列描述符内抢占索引的重新组装 (repopulation)。缺省地，如果下一抢占索引不是其自身权利内的线程索引，则通过从下一抢占索引所识别的调度层开始向下巡视调度层级来装该下一抢占索引。一旦完成了该组装，则将所识别出的线程索引放入抢占索引字段内，并且将该线程从就绪队列中虚拟地弹出，即，该线程从就绪状态转变到弹出状态（见图 41）。通过对最适任线程加锁、将其标记为已弹出以及标记一事件将其发回 TSSM 以用于重新调度，可使虚拟弹出变得明显。

[0680] 在某些情况下，派出队列事件导致中断。如果重新组装的抢占索引包含有效线程描述符索引并且启用了中断，则对与该派出队列相关联的系统处理器的中断将在除以下情况的所有情况下被断言，在该例外情况下，启用了早期中断断言标志并且下一抢占索引是线程，其中所述中断已经被 TSSM 断言。

[0681] 派出队列量度也被更新。

[0682] 虽然派出队列量度代表当前在运行的线程的适任性，但是要么派出队列被抢占，要么发生了派出队列的弹出。因此，要么执行中线程要被抢占（在此情况下，量度的更新在

最差情况下会有点过早），要么执行新线程，无论如何派出队列量度都将被覆写。

[0683] 在发生了篡夺现存的下一抢占索引的重新调度的情况下，必须将该现存的抢占索引虚拟地冲出回到就绪队列结构（见图 41）。在 TSOM 内，该线程被简单地标记为冲出并被推送回 TSSM 工作队列以用于处理。

[0684] 如果无法找到适任线程，则该操作简单地完成，客户机（处理资源）将在这些情况下空闲。

[0685] 设置派出队列适任性量度

[0686] 派出队列适任性量度反映了当前运行在由 ProcElementID 所索引的处理资源实例上的任务的执行优先级。然而，在某些优化下，派出队列适任性量度还可反映要开始执行的任务的执行优先级。

[0687] 派出队列优先级量度用于控制抢占，出于各种原因对其进行操作：

[0688] - 开始新任务

[0689] - 当前任务的完成

[0690] - 优先级倒置

[0691] - 功率管理

[0692] 在所有情况下，目的是要调节对就绪队列进行调度的操作。在更新导致修改池根节点的情况下，将该节点标记到 TSSM，以用于重新调度操作。对于池参与者和非池参与者内的量度传播的更多细节，请见上文。

[0693] 作为一种优化，当处理资源实例将当前在执行的线程推送回 SystemWeaver 内的就绪或阻塞状态时，执行优先级被自动重置为缺省量度。

[0694] 方法接口

[0695] 除了针对派出队列链表的基本状态操作（取得和设置状态）之外，输出管理器还提供针对其接口的以下命令：

[0696] Automated executing metric update(C_TSOM_CMD_SERVICE_TIME_SLICE_EXPIRE)

[0697] 调用者：TSIF

[0698] 该命令针对所识别出的处理器 ID 自动修改派出队列内保持的量度的量度 0 的最低有效位。因为参数是处理器的标识而不是派出队列描述符自身，所以该命令最初巡视派出队列描述符链表以寻找适当的描述符。所述修改是简单地反转所述最低有效位，假设适当设置了量度字段的保留部分，则这具有这样的效果：不管优先级是升序还是降序，都降低执行中线程的优先级。

[0699] 下面示出了参数：

[0700]

unsigned char ProcID	必须更新其执行优先级的处理器的处理资源 ID。

[0701] Set dispatch queue metrics（设置派出队列量度）

[0702] 调用者：TSIF（来自明确的系统调用）

[0703] 该命令为特定派出队列设置执行量度。下面的参数存在于命令结构中：

[0704]

WMEIndex	派出队列描述符的索引。
Metrics	所识别出的派出队列的新的执行量度。

[0705] Set default dispatch queue metrics(设置缺省派出队列量度)

[0706] 调用者:TSIF

[0707] 该命令将特定派出队列的执行metric(0)重置为缺省值(也保持在该派出队列内)。该函数没有参数。

[0708] Dispatch queue event(C_SCHED_ENTRY_DISPATCH_LIST)

[0709] 调用者:TSSM

[0710] 该命令在就绪队列的状态变化需要时使派出队列描述符被更新。

[0711] TSSM-调度管理器

[0712] 调度管理器对就绪队列结构的父子链接中固有的调度判定进行管理。

[0713] 架构

[0714] TSSM排他地从其工作队列接口得到命令供给,并且是纯事件驱动的。然而,特定行为是几个命令所公有的,下面描述该行为:

[0715] 重新调度

[0716] 重新调度函数根据调度层级中的一定义点重新评估调度判定。该重新调度操作是工作守恒的,除与层级的其状态可被一事件可设想地影响的部分相关联的工作之外不进行其他工作。

[0717] 重新调度函数有一个特别有趣的参数:UpdateRequired。UpdateRequired用于在调度器层操作之间传播更新状态。例如,虽然仍然必须管理其他状态,但是不对子层中的判定进行更新的弹出操作不需要引起父层内的整个调度循环。在此情况下,UpdateRequired为假。

[0718] 图60示出了重新调度操作的基本流程。层内调度器执行层内的调度,并导致层父亲头索引指针的更新。层间调度器面向派出节点对调度层级内的连续层进行缩放(scale)。要注意,用父索引调用层间调度函数,从而立即缩放层级的级别。池层间调度是特殊情况的调度算法,其是从单个节点(池根节点)扇出到多个节点(池静态节点)的唯一算法。

[0719] 该章节的其余部分描述调度算法的操作。

[0720] 层内调度器

[0721] 最基本的调度操作是遍历调度层内的元素的链表以识别最适任的,并据此更新父亲的头指针。

[0722] 在普通情况下,在将适当描述符推送至空调度层的情况下,无条件地更新父亲的头指针和元素的数量,并且将量度从新孩子有条件地传播(根据量度传播操作符)到父亲。

[0723] 图61示出了推送操作的更一般的情况。当前选项以及候选选择的有效性是以下多个因素的组合:

[0724] -该选择必须具有可被调度的内容。对于线程,这总是为真,但是对于调度器,这取决于其从属层级的内容。

[0725] - 如果该描述符是调度器，则其一定不能是休眠的，也一定不能设置其无效选择标志。

[0726] - 如果该描述符是标准线程，则其一定不能被加锁，其也一定不能位于工作队列中。

[0727] - 如果该描述符是标记线程，则父亲的总依赖计数值必须为零，并且该标记线程必须是层中剩下的唯一条目。

[0728] 要注意，候选仅与当前选择（前一调度操作的最适任描述符）进行比较，如果该候选项击败获胜者，则其一定是新的获胜者。如上所述，“调度对”与父调度层内保持的算法相关。“更新父亲”变量将这样的指令带回调用者，即，作为该操作的结果，此层的父亲也应被更新。

[0729] 在重新调度的一般情况下，例如，在已更新了量度的情况下或者在发生了弹出的情况下，必须重新评估整层以寻找新的最适任候选项。如图 62 所示，该处理运行图 61 的操作多次，直到已重新评估整层为止。

[0730] 层间调度

[0731] 层间调度在每个调度器层运行一次，其针对每个调度事件可能达到几次。层间调度高度依赖于父亲的类型。总而言之，层间调度持续调用层内调度，直到父亲变为派出队列为止。这有一种例外，该例外是遇到池分布节点的情况。

[0732] 图 63 示出了层间调度的基本流程。存在与派出队列描述符 (DQD) 父亲以及池根节点 (PRN) 父亲相关联的唯一行为。然而，在其他情况下，层间调度器简单地用当前父亲来替换当前子索引（从而沿调度层级向上叠代）并重新调用其自身。

[0733] 图 64 示出了层间调度例程中的派出队列处理。首先，根据在派出队列描述符 (DQD) 中定义的算法，针对在调度器根节点内保持的量度，来调度包含在 DQD 内的执行量度。如果该操作确定需要抢占，则根据启动重新调度操作的事件的类型对 DQD 下一抢占索引进行更新。如果该事件是线程推送，则将被调度的线程索引直接放置在下一抢占索引字段中，否则使用调度器根节点索引。然后指向 TSOM，叠代调度层级以寻找该线程描述符。

[0734] 池层间调度

[0735] 池层间调度用于识别应该选择哪个（如果有的话）相关联的处理资源实例为在当前池内保持的线程描述符提供服务。在该意义上，池层间调度以独特的方式工作，因为与这里描述的所有其他调度算法不同，其通常寻找最不适任的候选项。

[0736] 图 66 示出了在图 65 中用图形表示的池层间调度操作的流程。最初的层内调度操作确定来自该调度层级的候选是否比执行中任务中的任何一个更适任，如池分配层内的池静态节点内保持的量度所示。有两种结果，对是否需要更新的指示以及对必须进行更新的节点的识别。

[0737] 该算法然后继续进行，从由池根节点的适当的“next（下一）指针”所指示的池静态节点开始，叠代遍整个池分配层。

[0738] 对于每次循环，除了管理维护函数（诸如维护线程计数器和其他状态）之外，还针对池附加层调用层间调度。作为正常的情况，继续层间调度以沿着调度的整个层级向上传播，直到到达派出节点为止。层间调度采用参数“update required（需要更新）”，其指示其他调度层是否应该完全重新评估该调度判定。在池间调度的环境中，在两种情况下设置该

标志：

[0739] - 当前在处理的池静态节点是这样的节点，该节点被池内调度识别为用于处理池根节点之下的最适任线程的最适任节点。

[0740] - 池根节点之下的层级没有任何适任调度候选项。

[0741] 池层内调度

[0742] 图 67 示出了池层内调度的流程。在第一个实例中存在一种优化，其减少推送单一操作的调度时间，这提供了系统对于新任务变为可用的响应性。

[0743] 假设这不是排他推送操作，则调度操作将当前选项和候选项设置为池内的前两个 PSN 节点。该算法然后进入绕整个池分配层的循环。对于每次叠代，针对候选调度当前选择。要注意，对于池分配层和其他调度层使用相同的调度算法，然而，所选择的个体算法可能不同，这是因为在此情形下，正确的选项根据正常标准表现出最不适任量度。

[0744] 如果候选主管 (preside over) 当前选择，则将当前选择更新为候选，将候选更新为该层中的下一条目，并且处理继续直到候选变为 PRN 为止。

[0745] 在排他的推送操作中，避免了该叠代。将当前选择和更新节点简单地设置为该 PRN（其定义了当前调度选择）的父亲，并且候选是该 PRN 自身。

[0746] 在所有情况中，然后检查池根节点以寻找可调度内容，如果没有，那么设置“不更新”状态，并且该算法返回。然而，如果存在可调度内容，则处理继续进行至第二阶段，由此针对 PRN 自身调度来自 PRN 的现存选择（在排他推送操作中）或叠代的结果（在其他情况下）。如果 PRN 在该竞赛中胜出，那么需要更新，否则不需要。

[0747] 示例

[0748] 图 68 示出了与将线程描述符（节点 #5）推送进静态调度元素（节点 #3）相关联的 TSSM 调度器处理。第一个层内调度操作发生在重新调度函数的环境内并且是相对于父节点 #3 的。然后，重新调度沿层级向上移动一层，并且调用与父节点 #2 的层间调度。此叠代之后紧邻的叠代寻找父节点 #1，其是 DQD。因此，没有针对层内调度器的其他调用，并且在存储在根节点内的最适任候选项的量度和存储在派出节点内的执行中线程的量度之间进行调度比较。在此情况下，抢占是适当的，并且将派出队列事件发送至 TSOM。

[0749] 图 69 示出了更多级联的调度层级。这里，针对层级的附加层进行对层内调度函数的附加调用。但是要注意，当引起重新调度操作的调度事件在其范围之外时，调度层 5 和 3 保持不受影响。

[0750] 图 70 给出了第三示例，在此情况下为池调度操作。与以前一样，其中发生了线程推送事件的层要经受层内调度操作，在该示例中，新线程是池根层中最适任的。然后，池层间调度器进行对池层内调度器的调用，以确定任何系统处理资源是否应被新到达的线程抢占。在此情况下，池层内调度的结果是：在 WME 索引 7 处的与派出队列描述符相关联的处理资源实例应被抢占。

[0751] 然后，池层间调度循环遍池分配层，首先在节点 4 上调用层间调度。继而层间调度调用层内调度以更新调度器层 1，但是因为这不是被抢占的层，所以不需要穷举调度，将处理限制为维护状态信息，因此，没有到达 TSOM 的派出队列事件，也没有到达系统处理资源的中断。

[0752] 下一调用是层 2 上的层间调度。在此情况下，适当地调度该层以确立新推送的线

程描述符是否比任何其他候选项更为适任。最终将候选项量度与存储在派出队列描述符内的执行中线程的量度进行比较，以确定抢占是否是适当的。在该示例中，相应地将派出队列事件发送至 TSOM 并且对系统处理资源实例中断加标记。

[0753] 再一次，仅仅重新评估推送线程事件的范围所及的那些调度层。

[0754] 方法接口

[0755] 排他地通过工作队列指向 TSSM。

[0756] Service thread descriptor event（服务线程描述符事件）

[0757] 调用者 :TSIF，TSPM，TSOM

[0758] 该命令的行为依赖于接收到的线程的标志内的设置 :

[0759] - 如果设置了推送或冲出标志，则父亲的总线程元素计数值增加。如果设置了弹出标志，则父亲的总线程元素计数值减小（要注意，以上的组合的净效果可能是空 (NULL))。

[0760] - 设置了推送标志的线程然后被链接到就绪队列层级。在其转变离开阻塞状态的情况下，依赖线程的父计数值也减小。

[0761] - 标记线程唯一的专有行为是将父亲解锁（将调度器层转变到“待释放”状态，见图 42)。

[0762] -TSSM 请求在处理该命令时释放在僵尸状态中接收到的任何线程描述符。

[0763] 该命令总是请求重新调度。

[0764] Reschedule pool distribution tier（重新调度池分配层）

[0765] 调用者 :TSOM

[0766] 该命令由于执行中的线程的量度改变而被调用。在派出描述符参与池的情况下，这些量度是通过分配层级而传播的并最终调用该函数以重新评估调度判定。

[0767] Reschedule or delete scheduler tier（重新调度或删除调度器层）

[0768] 调用者 :TSOM

[0769] TSOM 出于两种原因调用该命令 :

[0770] - 由于执行中线程的量度改变，其中派出描述符不参与池，重新评估抢占是否适当。该操作传递调度根节点。在调度层在其层级中包含线程描述符的情况下，请求重新调度。

[0771] - 为了删除调度层，其中 TSOM 确定这样做是适当的。该确定基于加锁的标志、依赖线程计数值以及子元素的数量（分别为假、0 和 0)。描述符的实际释放将其自身表示为对 TSIM 的请求。

[0772] Push an independent element（推送独立元素）

[0773] 调用者 :TSIF

[0774] 在初始化期间可使用该命令以将静态调度层推送进调度层级，或者可在运行时间期间动态地使用其来添加线程描述符或动态调度层级。

[0775] Update thread metrics（更新线程量度）

[0776] 调用者 :TSIF

[0777] 更新就绪队列层级内的线程描述符的量度。仅当线程描述符处于就绪状态（图 41) 时才可更新量度。该命令引起重新调度。

[0778] Update scheduler state（更新调度器状态）

[0779] 调用者 :TSIF

[0780] 该命令使得能够更新调度器算法、量度和量度传播算法。该命令引起重新调度。

[0781] Activate scheduler tier(激活调度器层)

[0782] 调用者 :TSIF

[0783] 该命令激活休眠的调度器层。该命令引起重新调度。

[0784] De-activate scheduler tier(去激活调度器层)

[0785] 调用者 :TSIF

[0786] 该命令去激活休眠的调度器层。

[0787] TSMM- 存储器管理器

[0788] 存储器管理器(图 71)提供用于聚合对 TCM 的访问的复用 / 解复用行为。它还提供加锁能力以确保在多个子块之间共享的资源的完整性。

[0789] 架构

[0790] 从架构的角度,存储器管理器主要可被视为复用 / 解复用器,其聚合六个可能的请求者中对 TCM 的访问。它还在多个子块尝试访问相同的资源的情况下通过实现带锁的高速缓存来保持 WME 的完整性。

[0791] 访问聚合

[0792] 访问聚合由调度器控制。该调度器是非对称的 :

[0793] -TSIF 具有最高优先级

[0794] -TSOM 具有次高优先级

[0795] - 所有剩余请求者具有相等的优先级,并按工作守恒的轮转方式对待。

[0796] 带锁的高速缓存

[0797] 每个块分配一至四把锁。这些数量表示子块请求者可排他访问的 WME 的数量。阻塞请求已加锁资源的子块,直到该资源变为可用。多个块之间针对同一资源的竞争通过优先级解决。

[0798] 调度序列图

[0799] 推送事件

[0800] 图 72 中的序列图示出了推送事件之后的子块间交互。要注意,引入工作队列来存储命令,因为 TSSM 和 TSOM 自身都是单线程的。请参照图 68 来获得代表性调度层级。

[0801] 推送事件之前的状态为,线程 #4 是派出队列描述符 #1 内的当前抢占索引。由于派出队列描述符 #1 被推送进 TSOM 工作队列,第一次重新调度识别出线程 #5 比线程 #4 更适任。

[0802] TSOM 工作队列中的派出队列描述符引起 TSOM 内的派出队列事件。该事件虚拟地弹出线程描述符 #5 并相应地设置派出队列量度。该弹出操作引起就绪队列的状态改变,因此必须调用重新调度。这是通过将线程 #5 推送进 TSSM 工作队列同时设置弹出标志而实现的。

[0803] 因为线程 #4 先前已虚拟地弹出,所以现在需要被冲出回到就绪队列结构。这还构成就绪队列结构内的状态改变,因此需要另一次重新调度。这是通过将线程 #4 推送进 TSSM 就绪队列结构同时设置冲出标志而实现的。

[0804] 要注意,因为虚拟弹出的线程和冲出的线程可能位于就绪队列层级中大不相同的

部分，所以第二次和第三次重新调度操作无法合并。

[0805] 弹出事件

[0806] 图 73 中的序列图示出了当从“虚拟”派出队列弹出线程描述符时 TSIF、TSOM 和 TSSM 之间的交互。

[0807] 弹出命令自身是 TSIF 通过命令接口或系统互联而接收到的。TSIF 通过将派出队列描述符推送至 TSOM 工作队列同时设置弹出标志，而向 TSOM 发出派出队列弹出命令。

[0808] TSOM 工作队列中的派出队列描述符引起 TSOM 内的派出队列事件。派出队列事件处理器指示 TSSM 请求将刚被弹出的线程描述符（在此情况下为线程 #5）放回空闲链表中。然后由 TSOM 虚拟地弹出存储有用于执行的下一最适任候选的下一抢占索引。这表示就绪队列结构的状态改变，因此 TSOM 指示 TSSM 重新调度，将下一抢占线程推送进 TSSM 工作队列同时设置弹出标志。

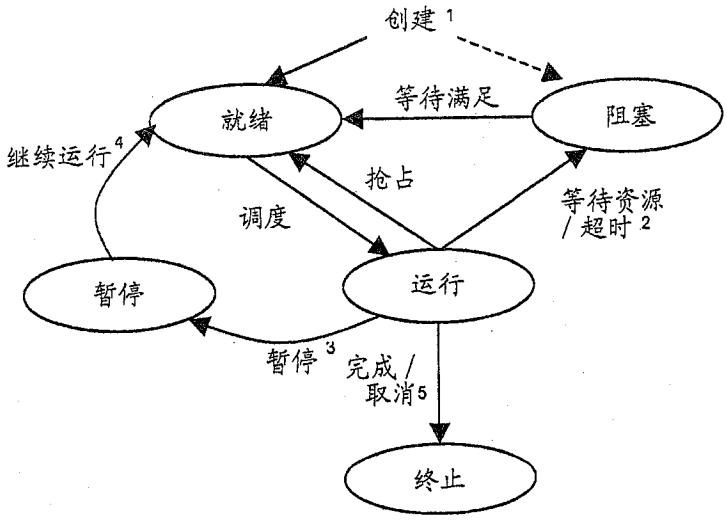

创建 1
等待满足
就绪
阻塞
继续运行 4
抢占
调度
等待资源
/ 超时 2
暂停
运行
暂停 3
完成 /
取消 5
终止

图 1 :任务状态图

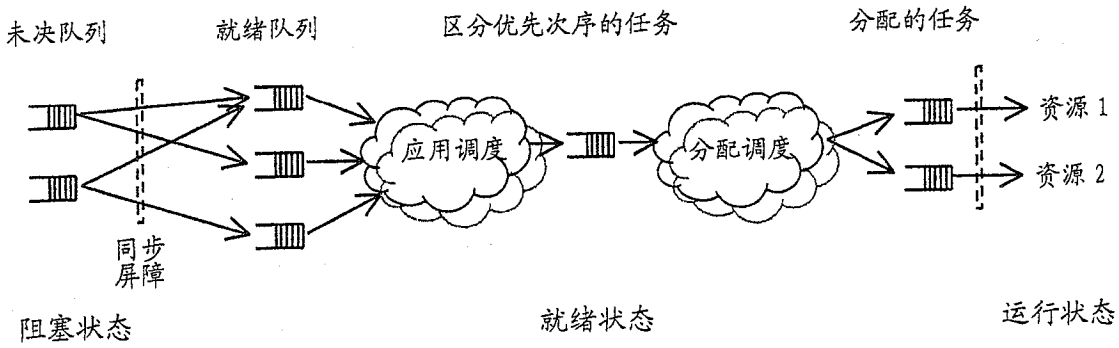

未决队列
就绪队列
区分优先次序的任务
分配的任务
应用调度
分配调度
资源 1
资源 2
同步
屏障
阻塞状态
就绪状态
运行状态

图 2

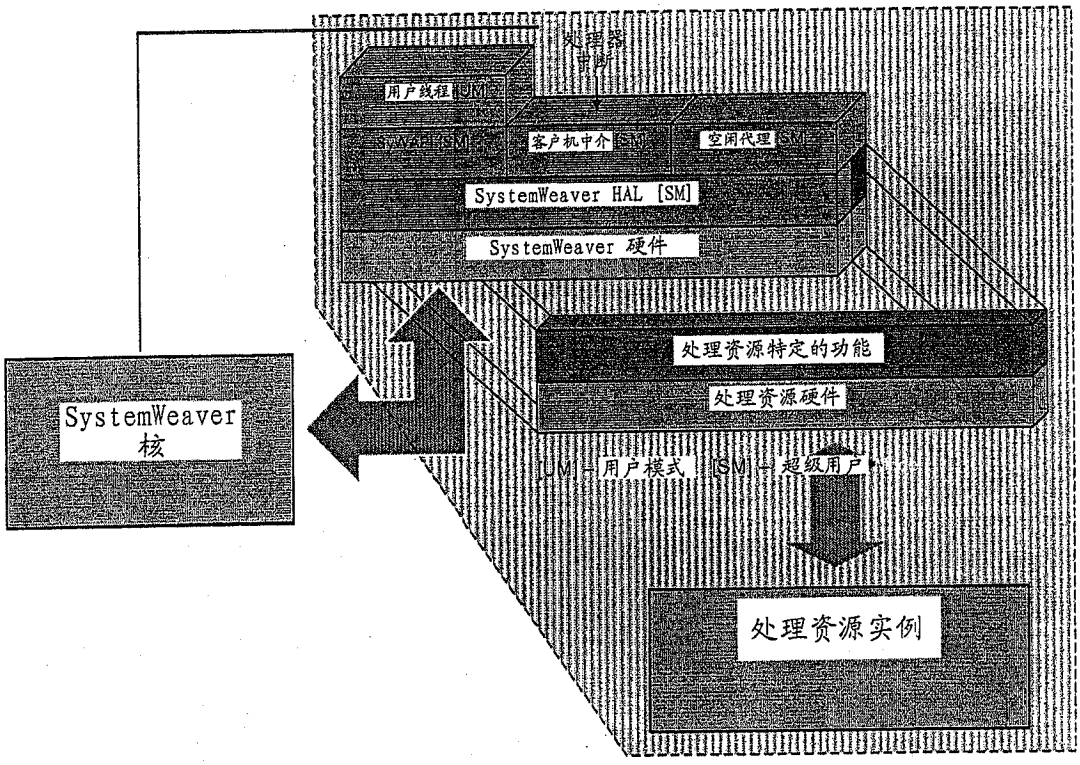


图 3 :线程管理和分配系统的组件

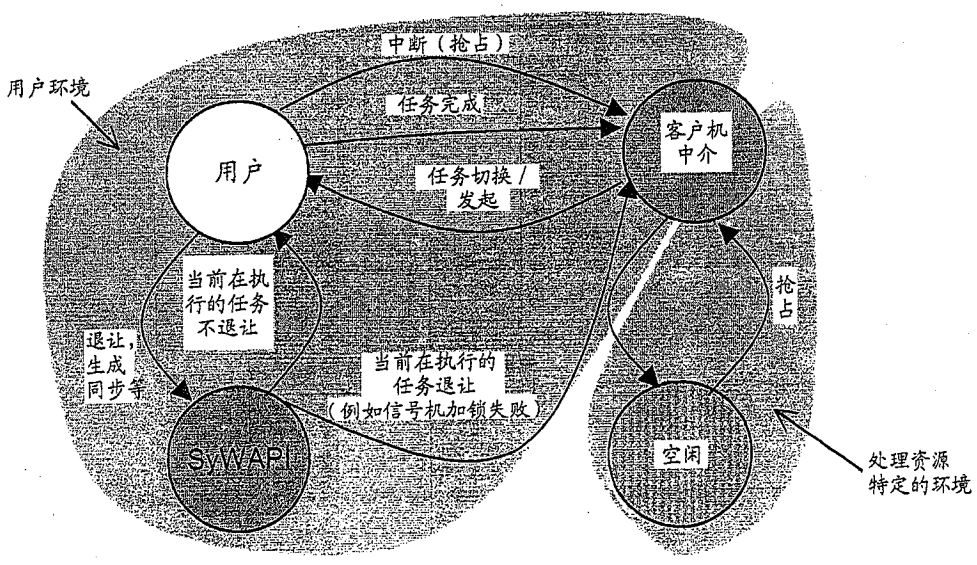


图 4 :线程管理和分配系统的客户机状态机

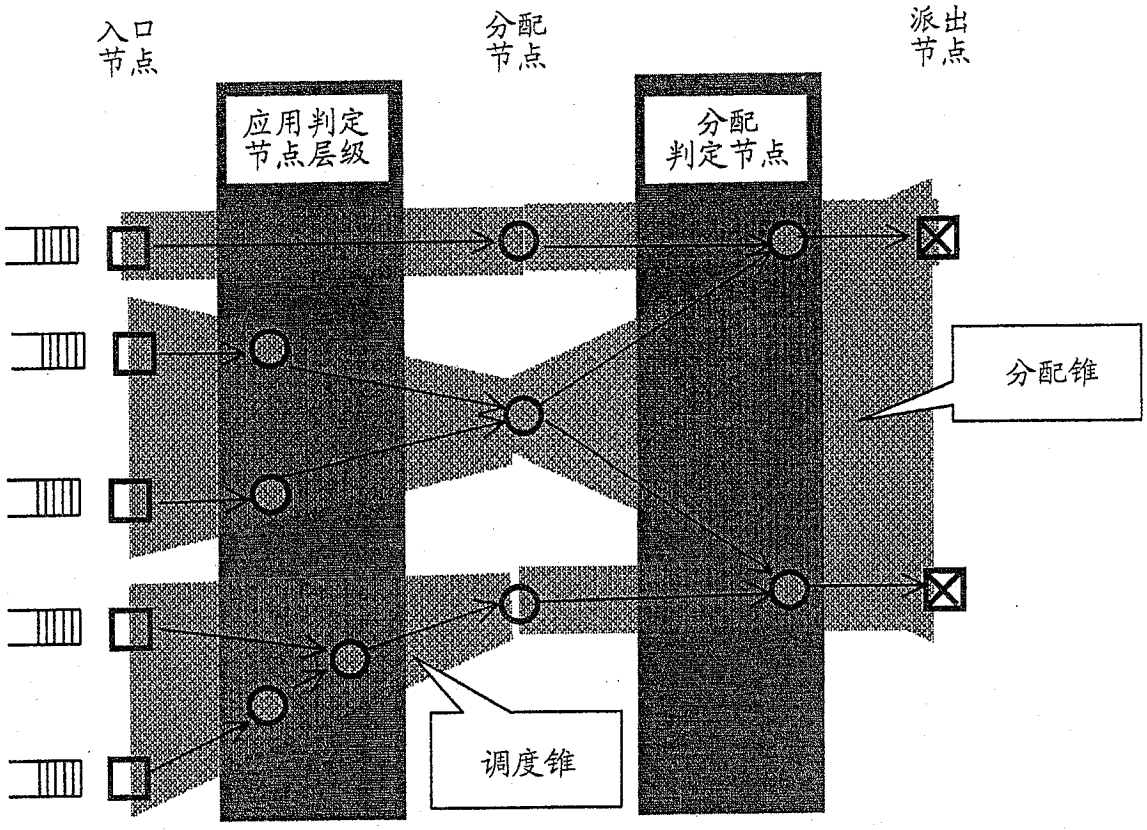


图 5 :概念上的调度结构

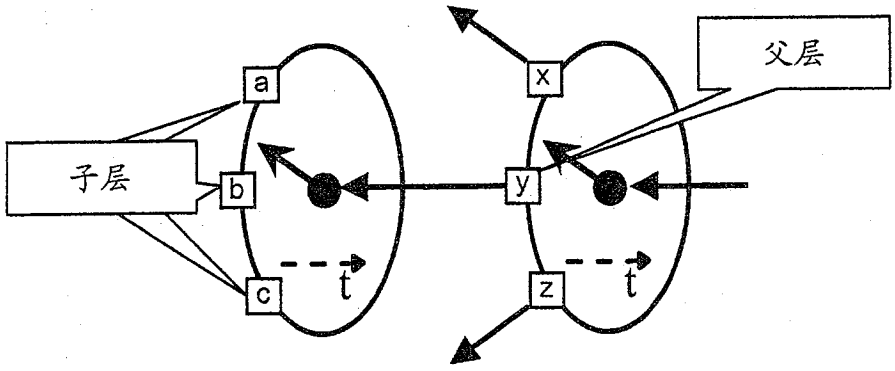


图 6 :父和子调度器关系

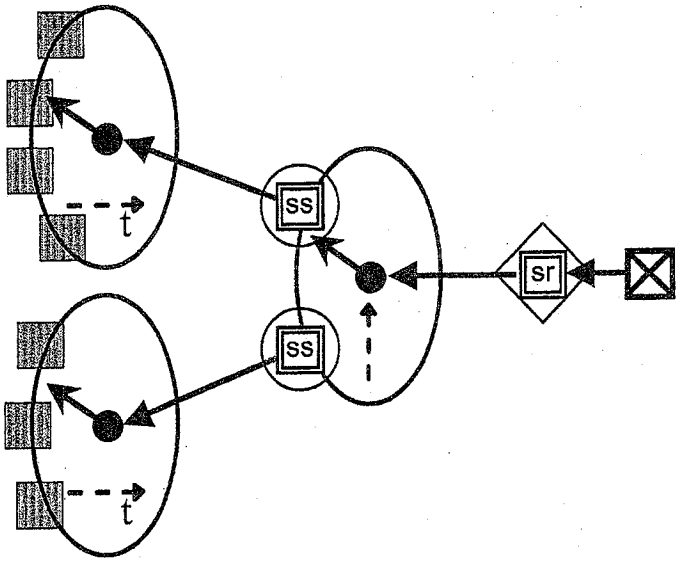


图 7 :基本调度锥

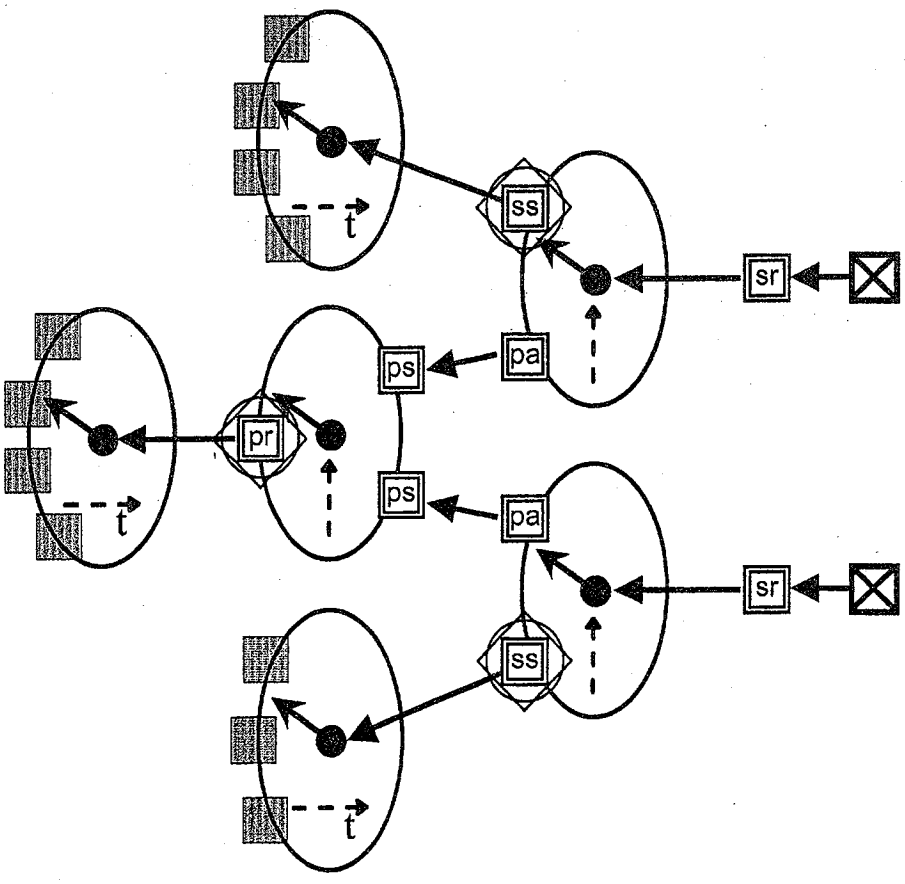


图 8 :一种典型处理资源池

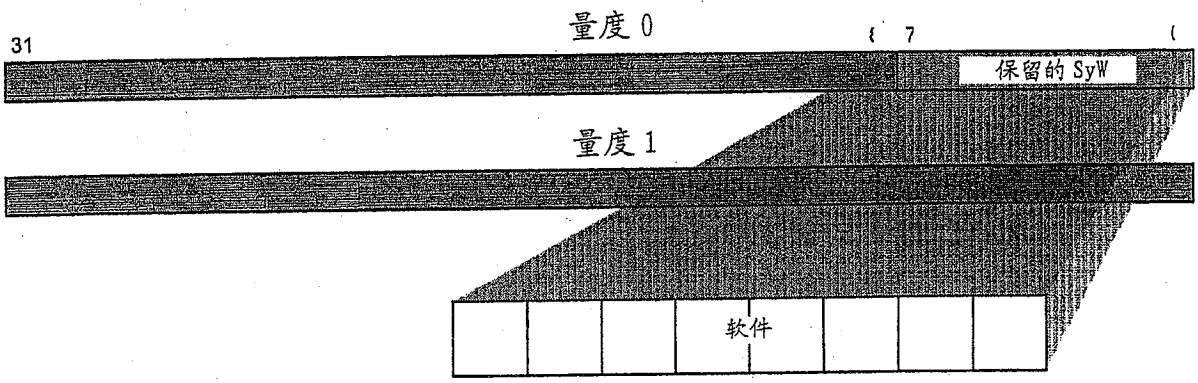


图 9：任务描述符量度

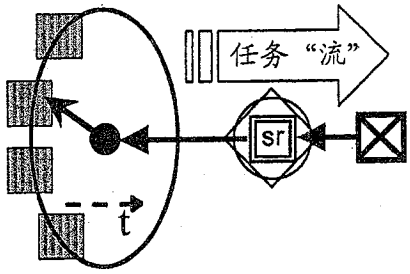


图 10

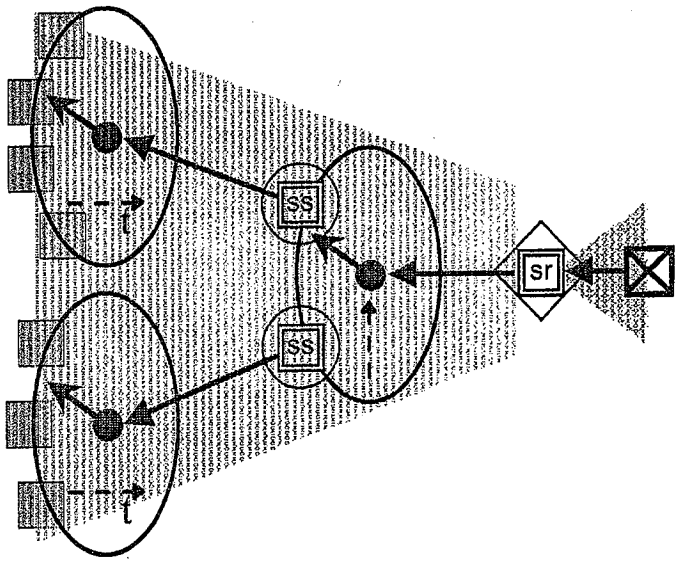


图 11：单个处理资源实例调度层级的示例

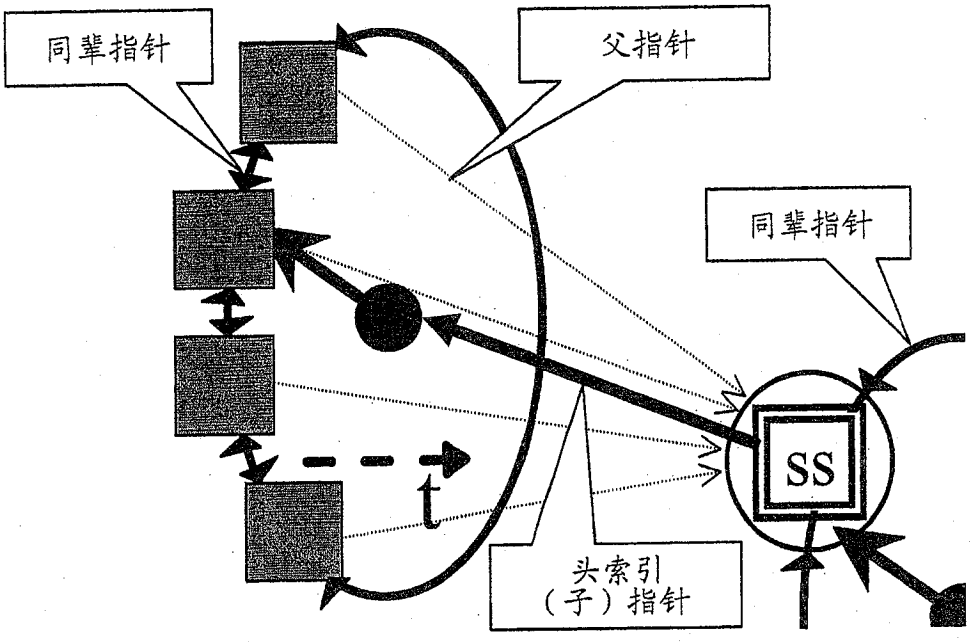


图 12 :调度层指针配置

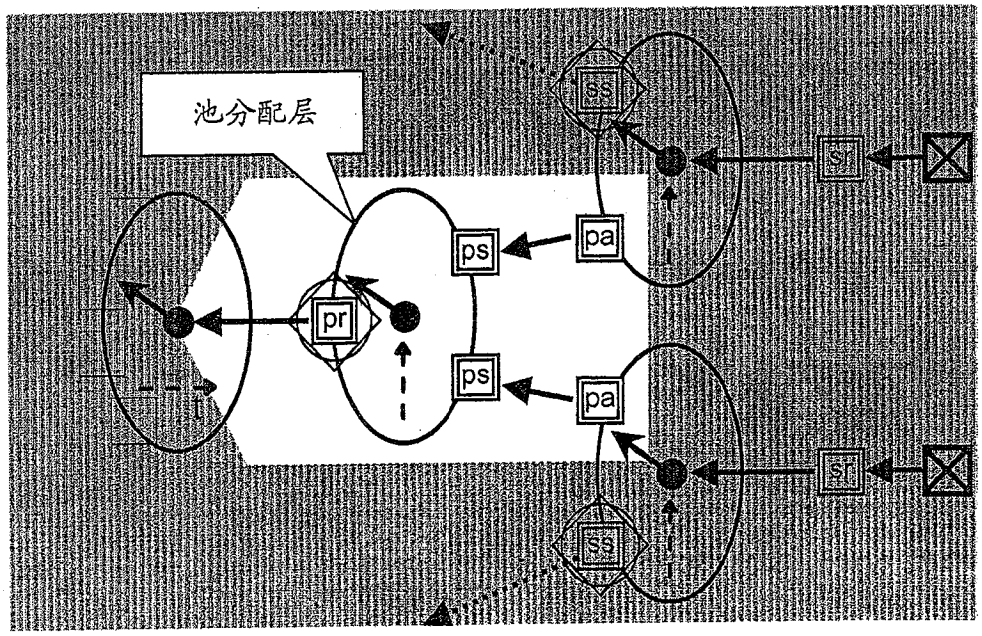


图 13 :池分配层结构

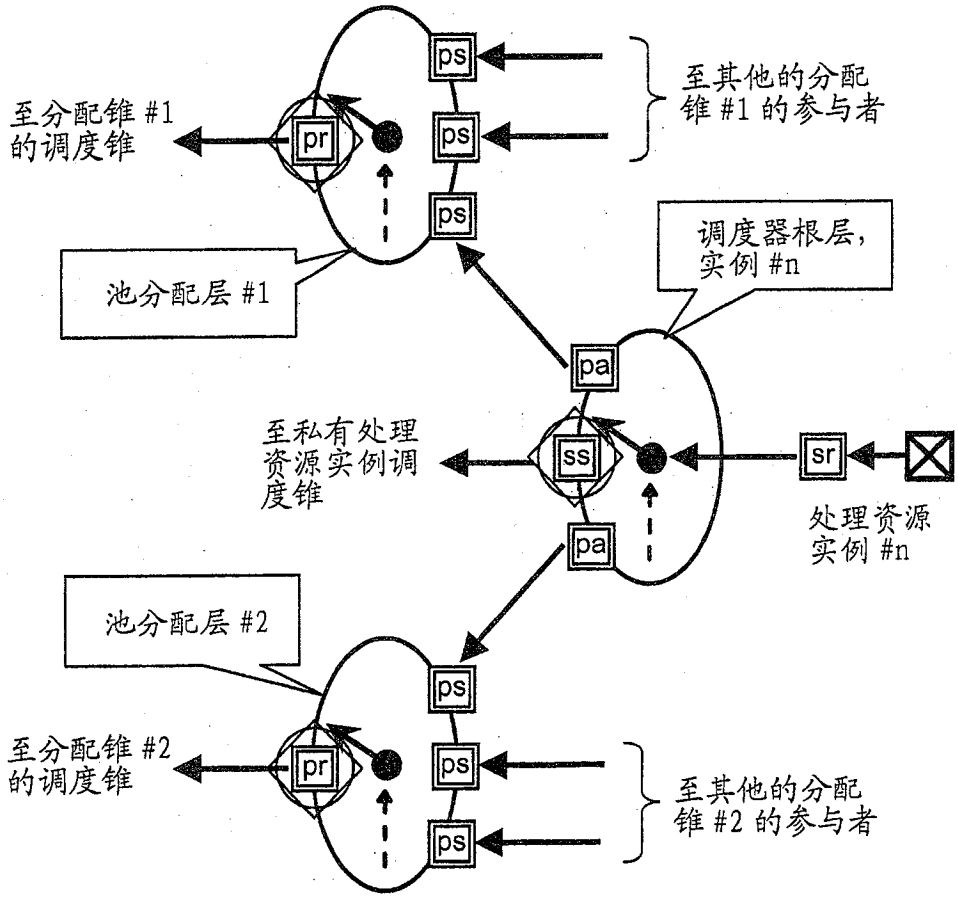


图 14 :参与多个分配锥的处理资源的示例

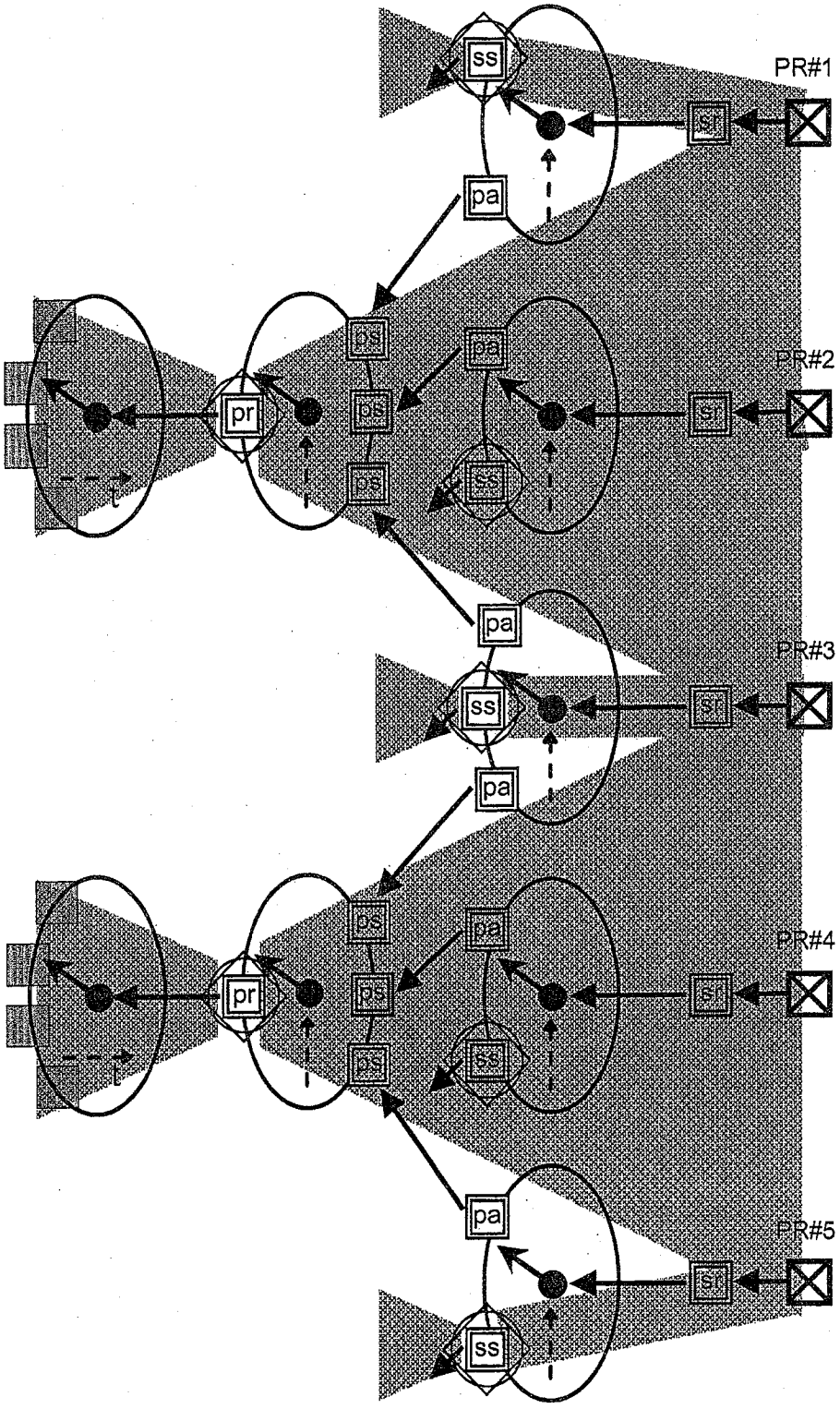


图 15 :一种具有两个分配池 的 5 处理器配置

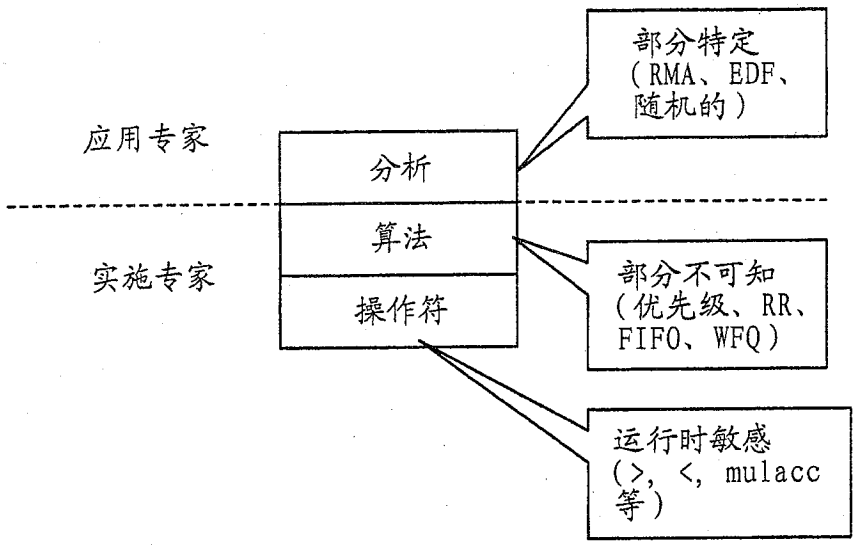


图 16 :调度分析、策略和操作符

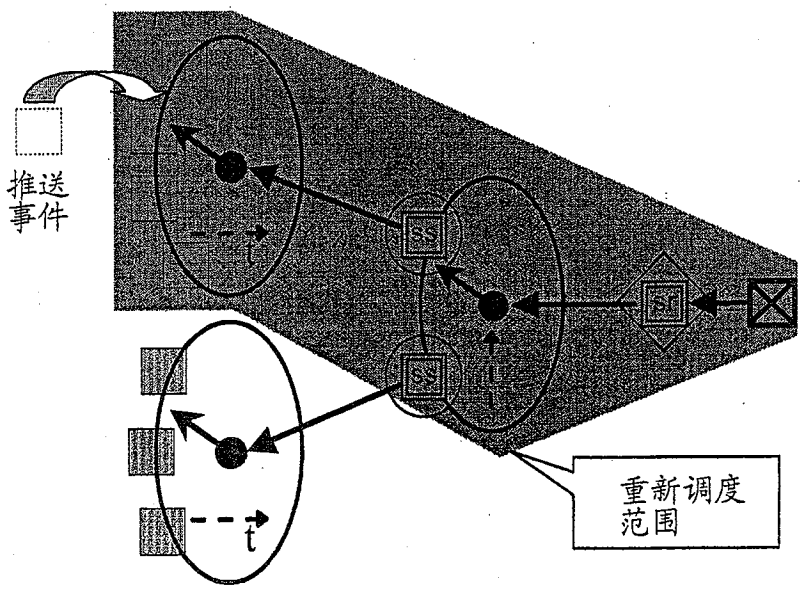


图 17 :简单推送操作的重新调度范围

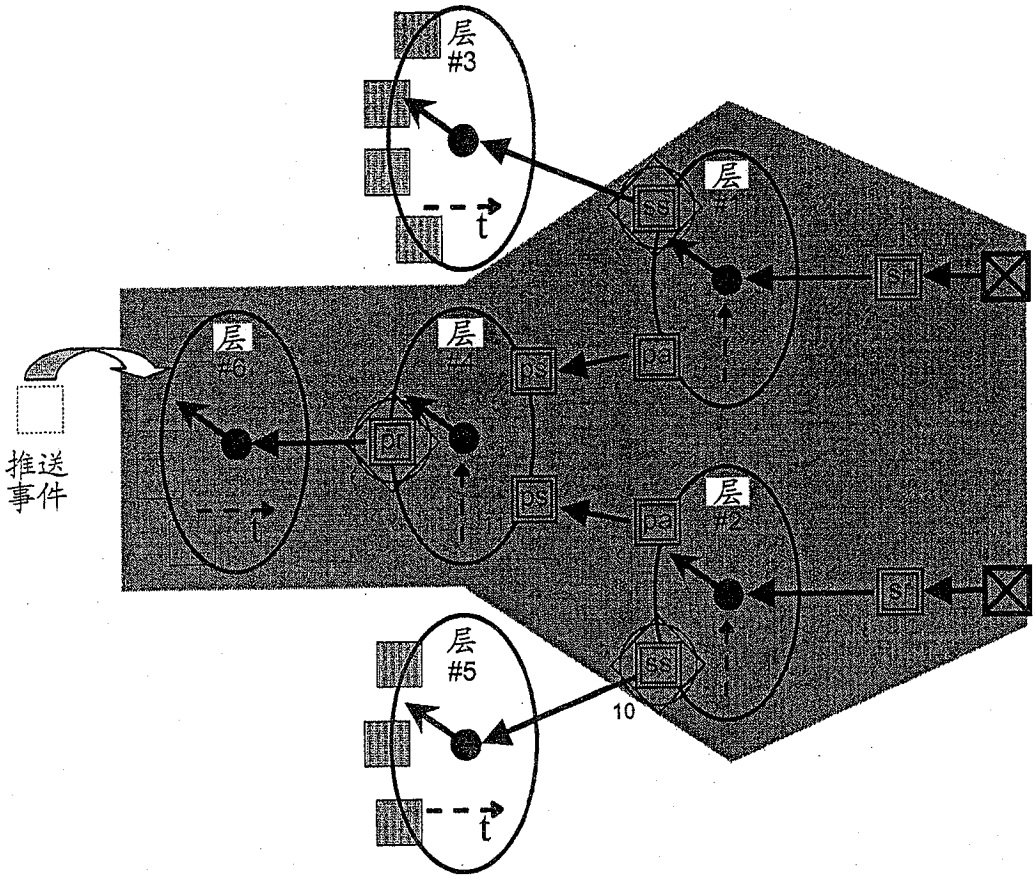


图 18 :简单调度和分配锥的重新调度范围

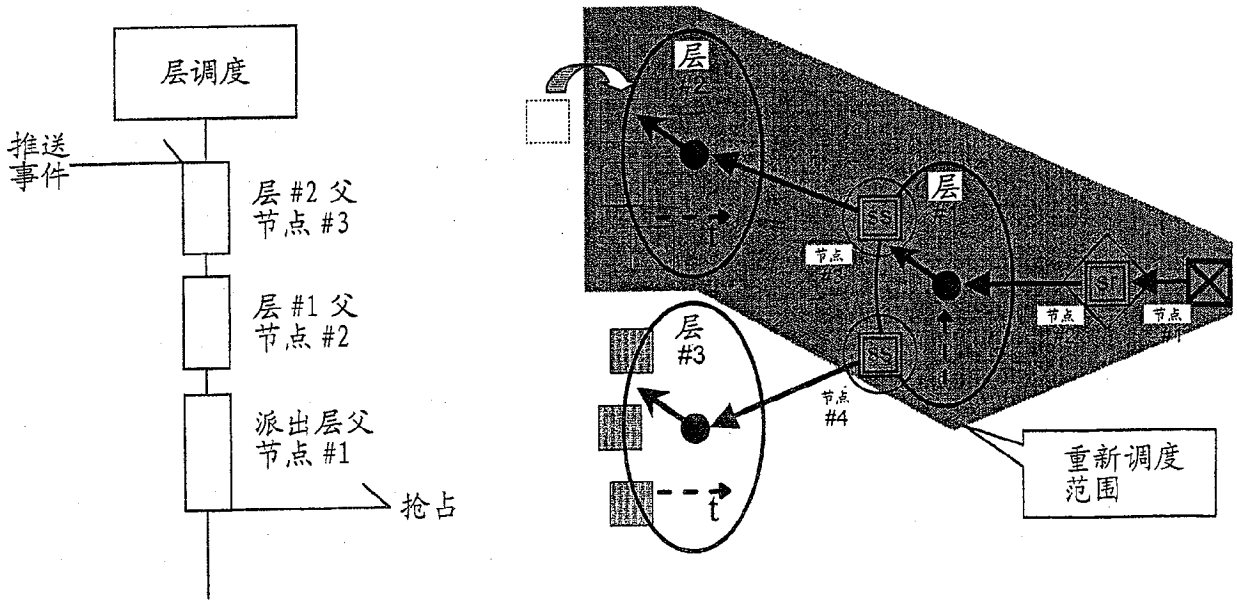


图 19 :简单推送操作的重新调度范围

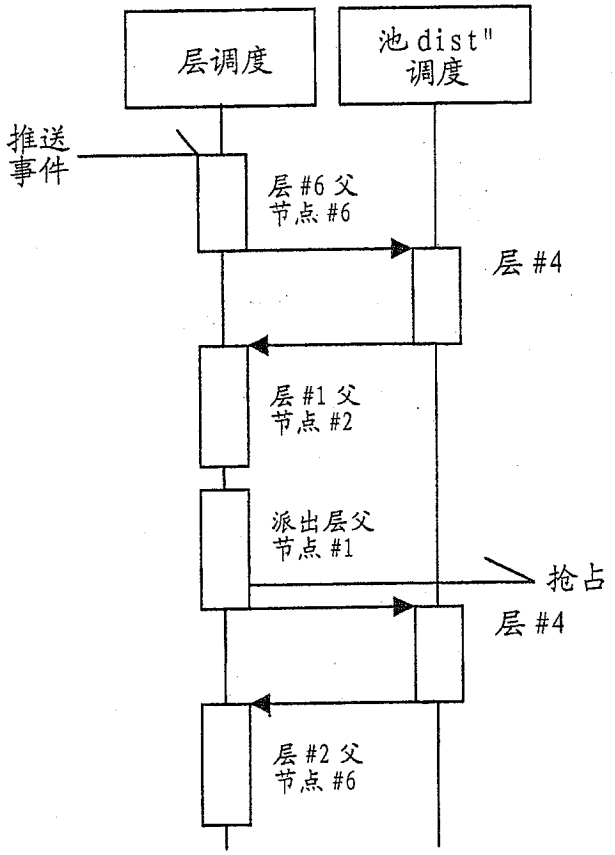


图 20 :推送到池分配节点 的重新调度序列

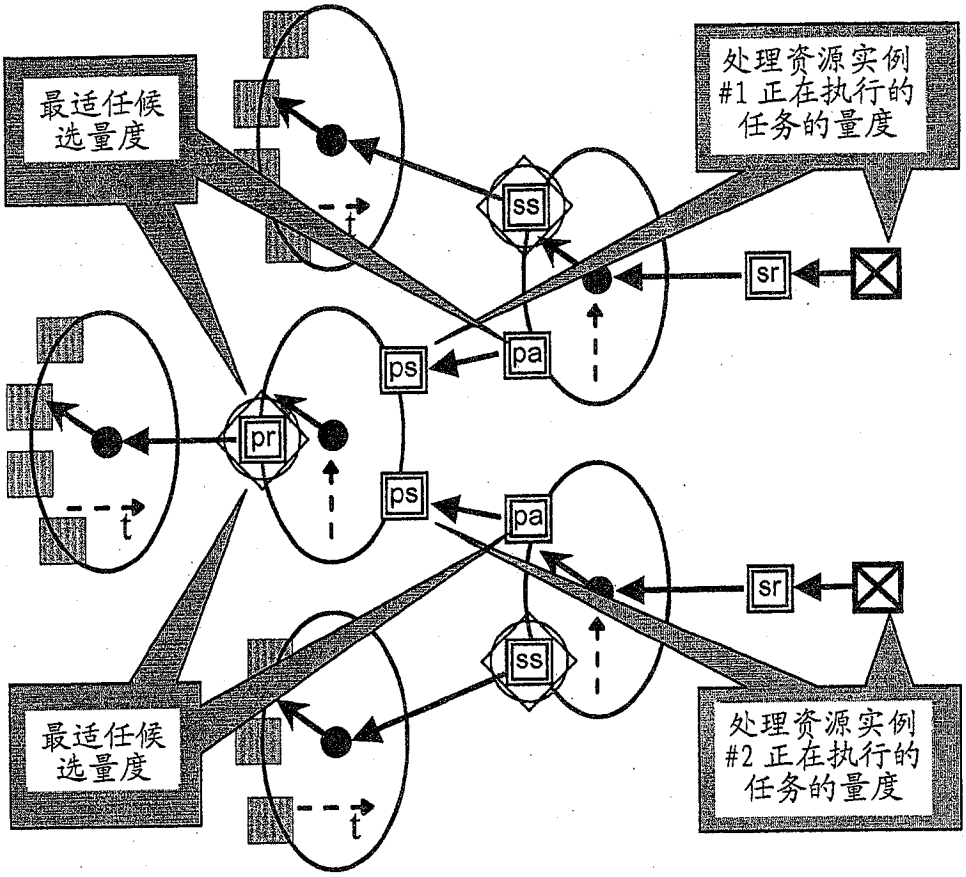


图 21 :处理资源池中的量度传播

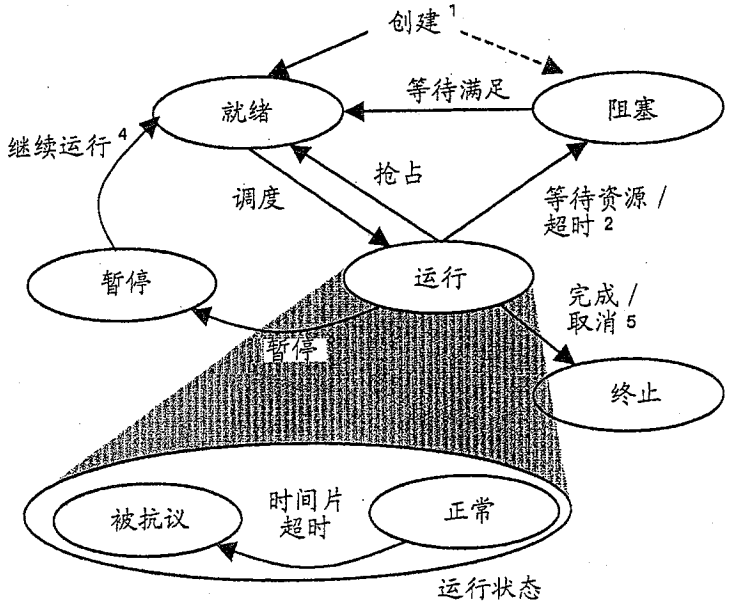


图 22 :任务状态图

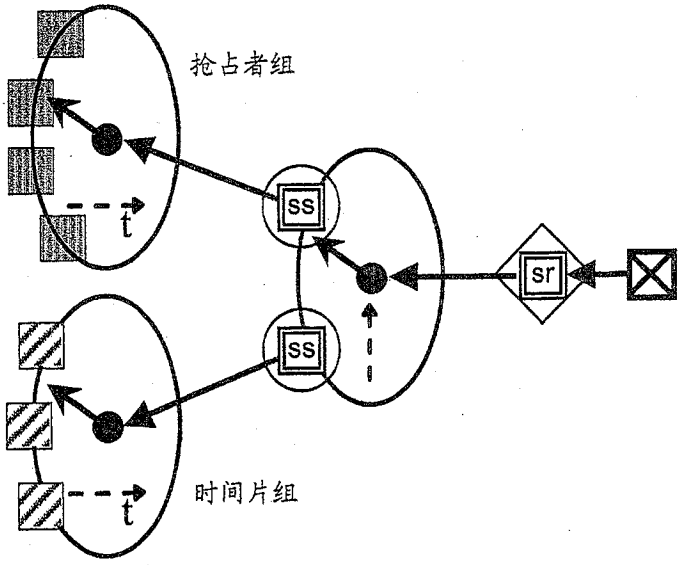


图 23 :示例时间片配置

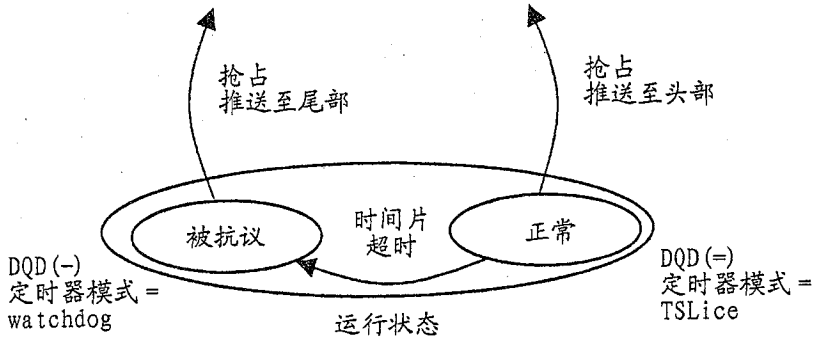


图 24 :任务优先级 delta 图

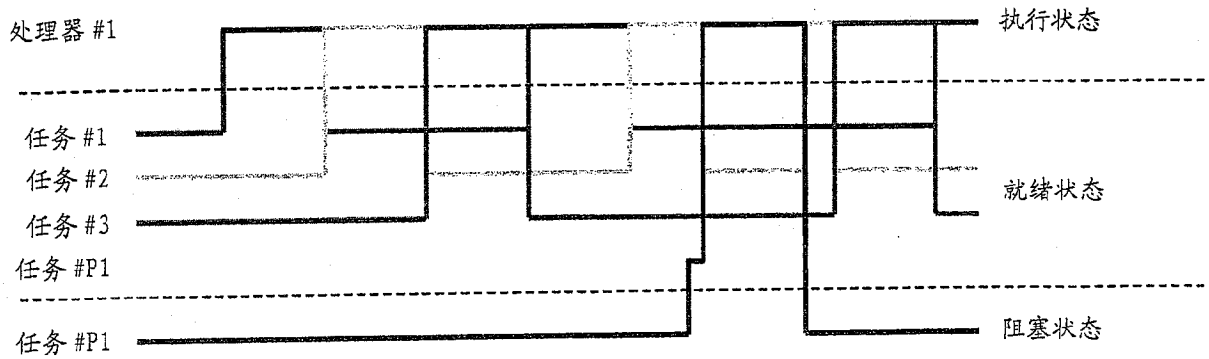


图 25 :单个处理器 3 时间片、1 抢占

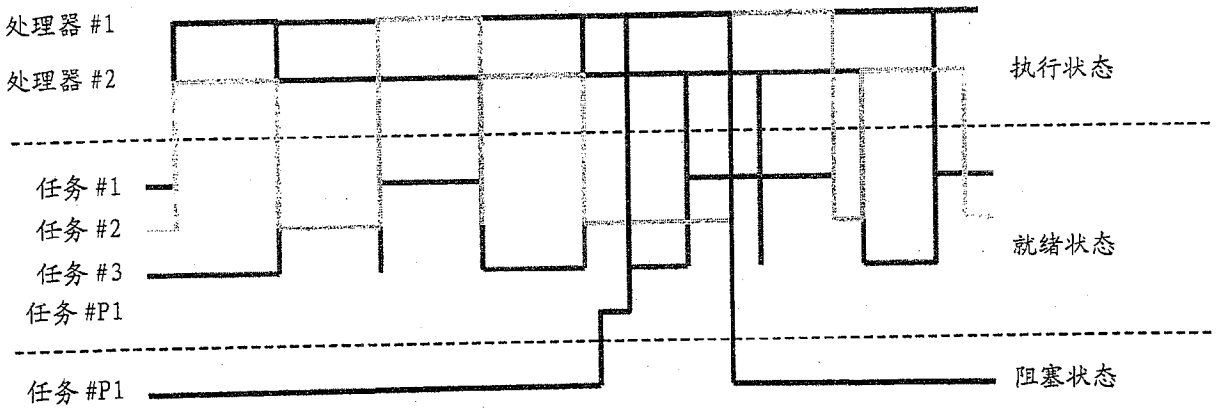


图 26 :双处理器 3 个时间片任务、1 个抢占任务

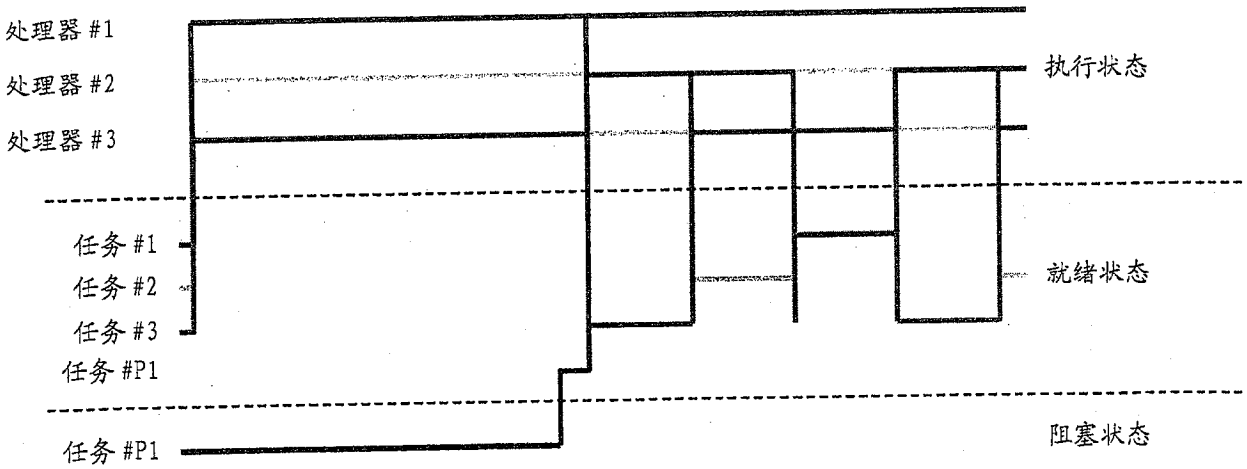


图 27 :三处理器 3 个时间片任务、1 个抢占任务

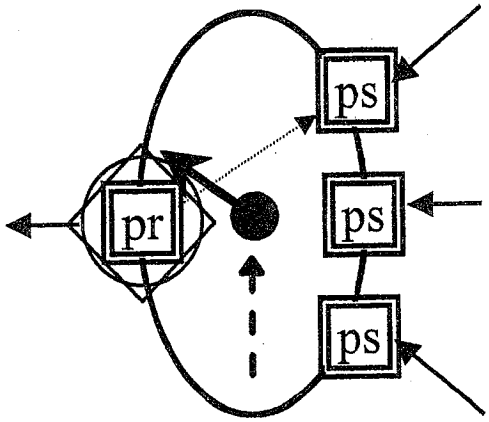


图 28 :分配层调度

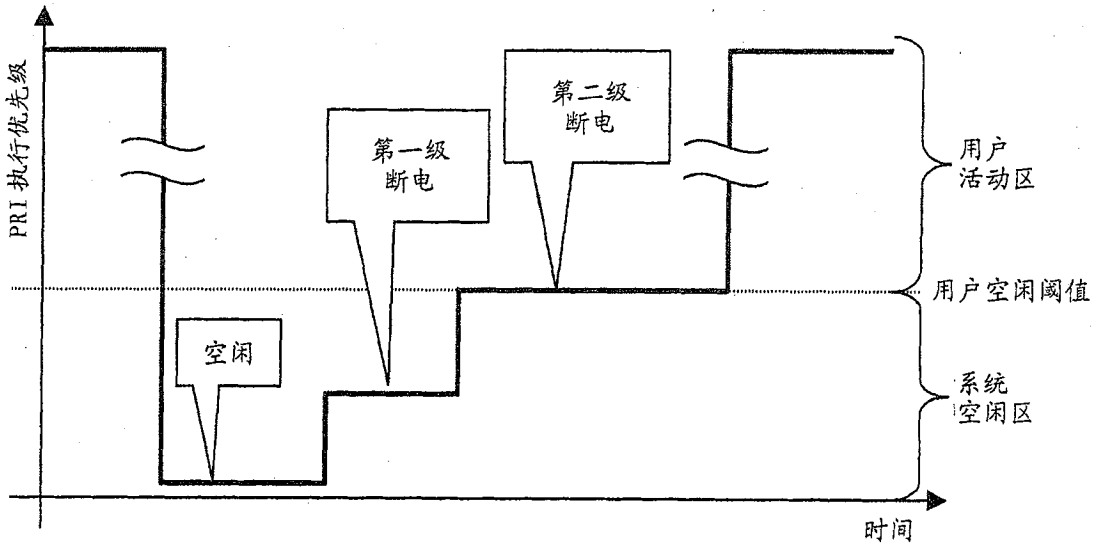


图 29 :示例客户中介断电

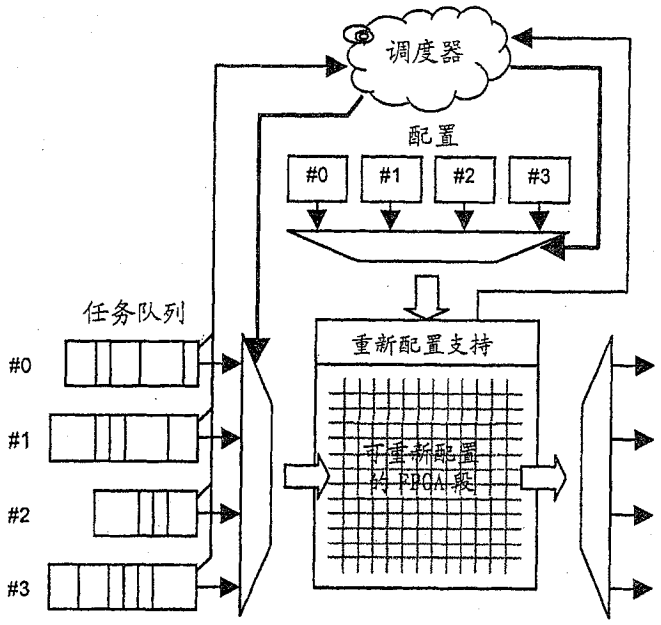


图 30 :重新配置管理架构

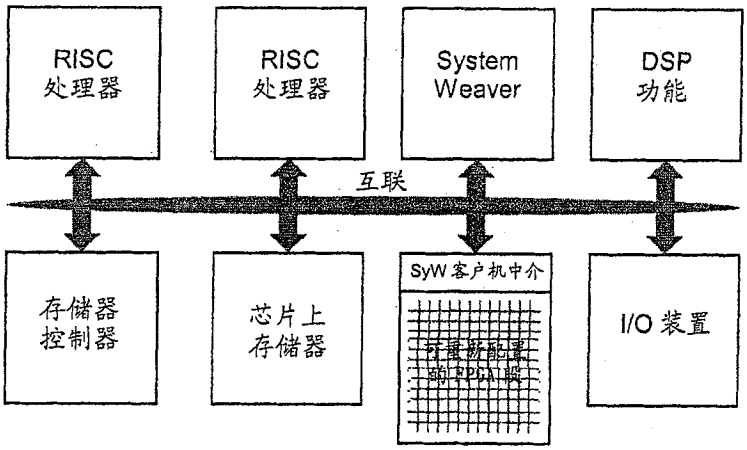


图 31 :实例 SystemWeaver 系统架构

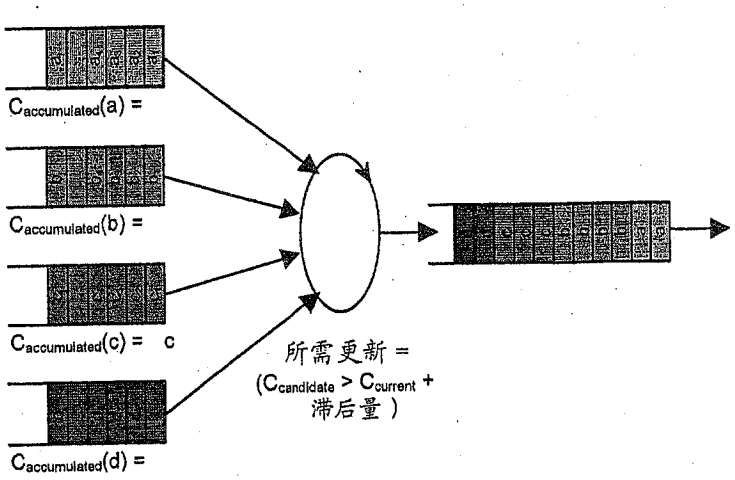


图 32 :最小环境颠簸 的调度器分组

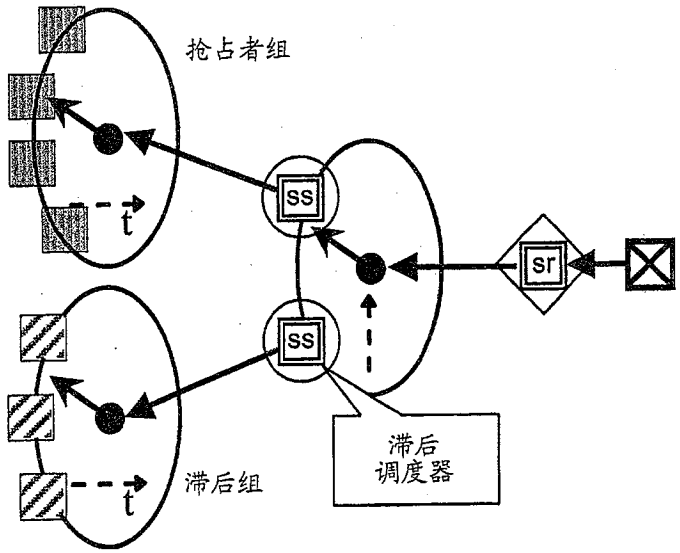


图 33 :示例滞后调度配置

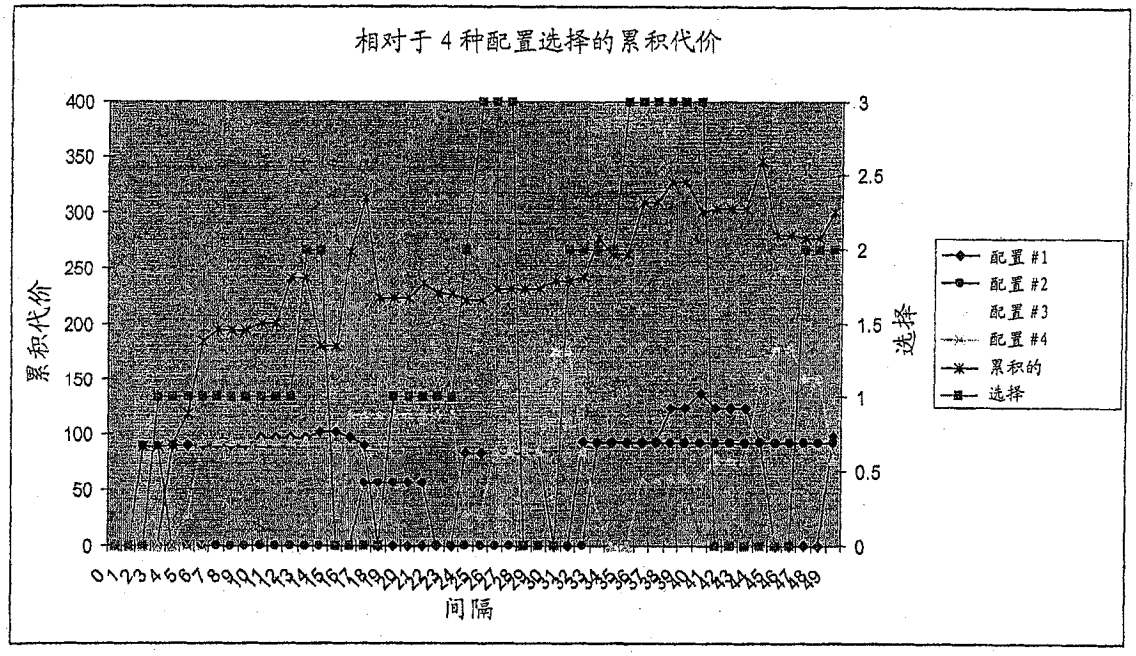


图 34 :基于滞后的调度的仿真结果

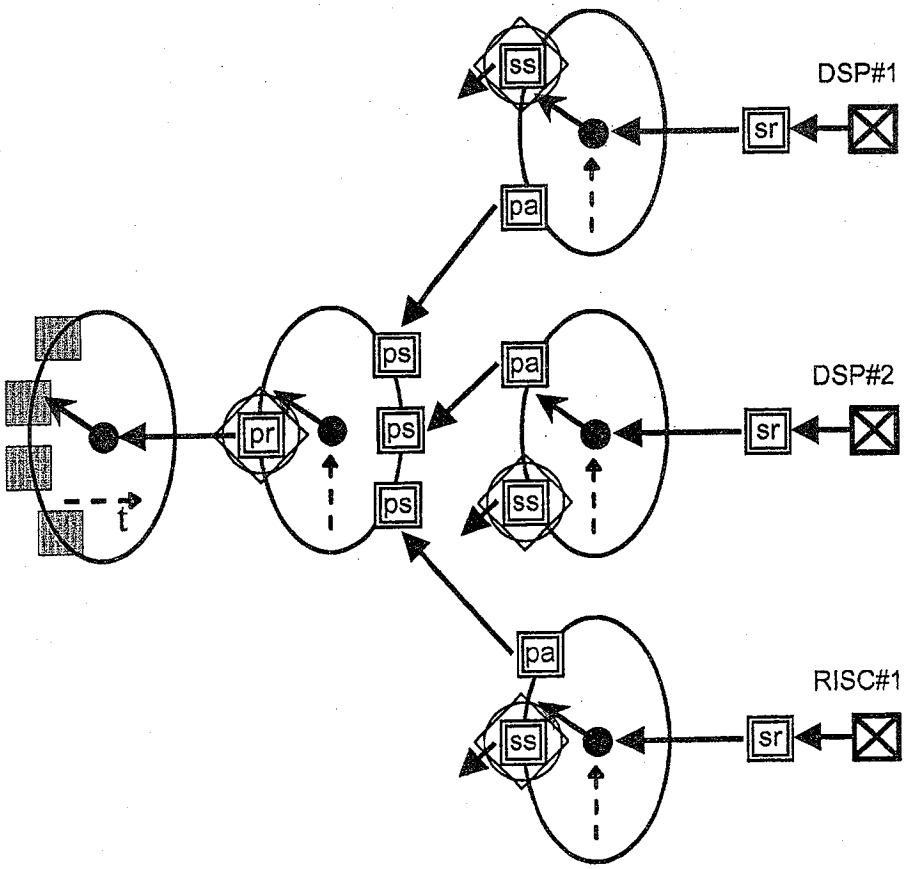


图 35 :复合调度算法示例

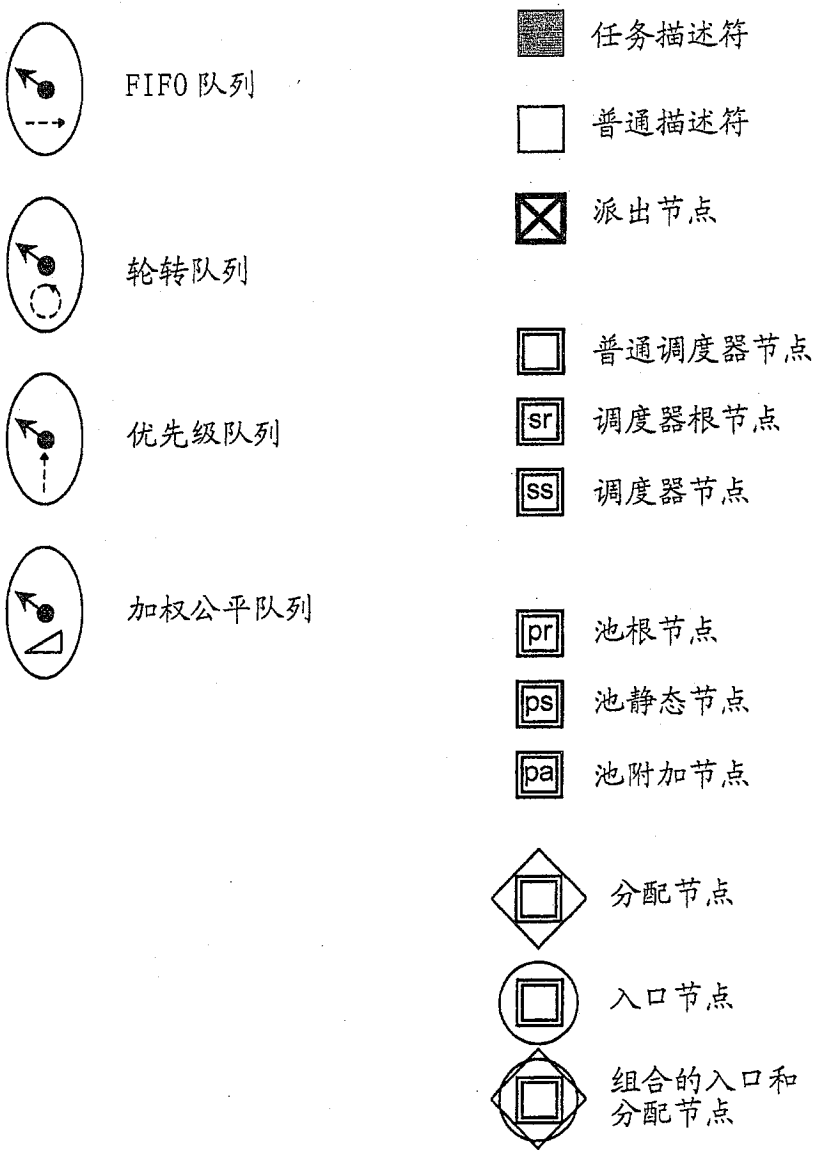

FIFO 队列
轮转队列
优先级队列
加权公平队列
任务描述符
普通描述符
派出节点
普通调度器节点
sr
调度器根节点
ss
调度器节点
pr
池根节点
ps
池静态节点
pa
池附加节点
分配节点
入口节点
组合的入口和
分配节点

图 36 :调度图的图例

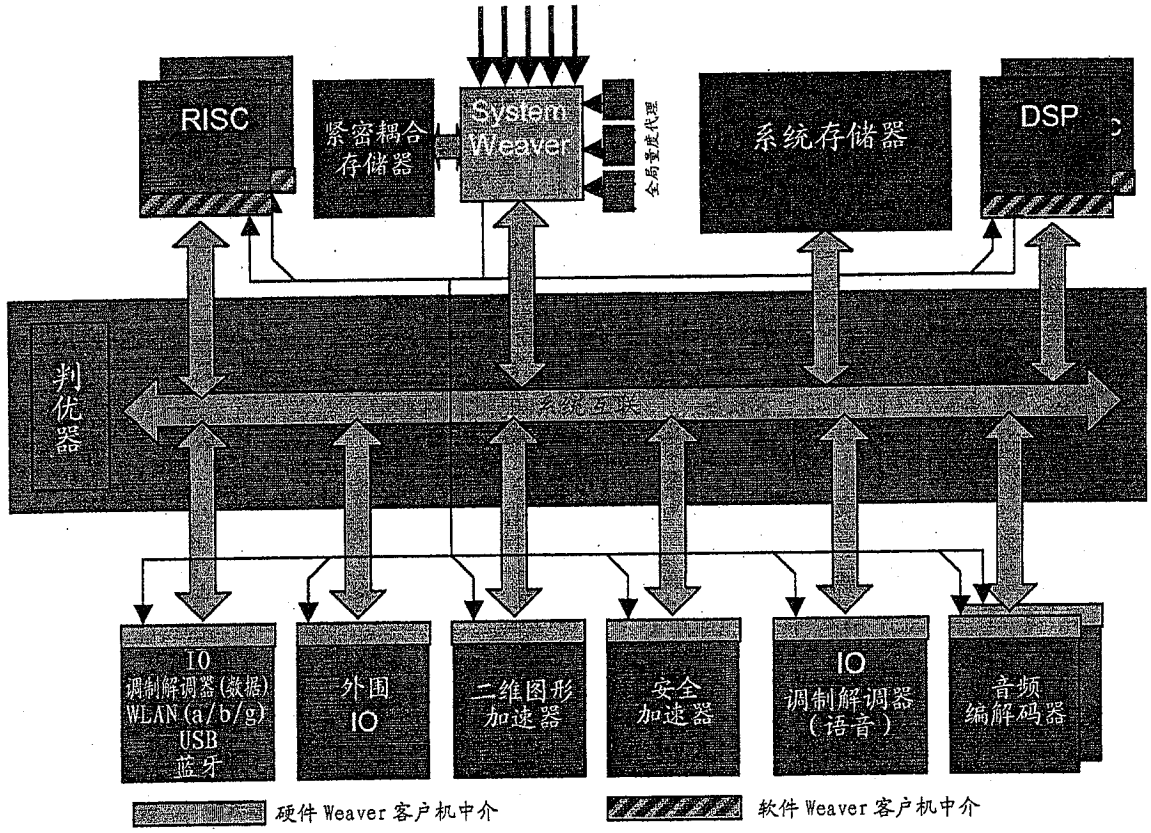

RISC
紧密耦合存储器
System Weaver
全局量度代理
系统存储器
DSP
判优器
系统互联
IO
调制解调器(数据)
WLAN(a/b/g)
USB
蓝牙
外围IO
二维图形加速器
安全加速器
IO
调制解调器(语音)
音频编解码器
硬件 Weaver 客户机中介
软件 Weaver 客户机中介

图 37

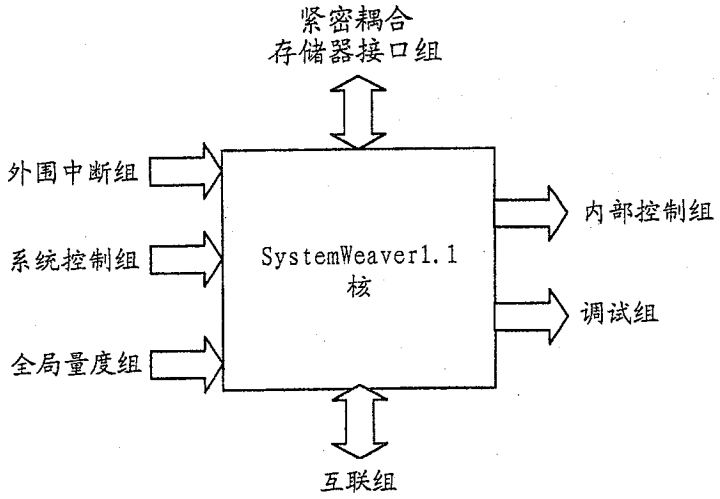

紧密耦合
存储器接口组
外围中断组
系统控制组
全局量度组
SystemWeaver1.1
核
内部控制组
调试组
互联组

图 38

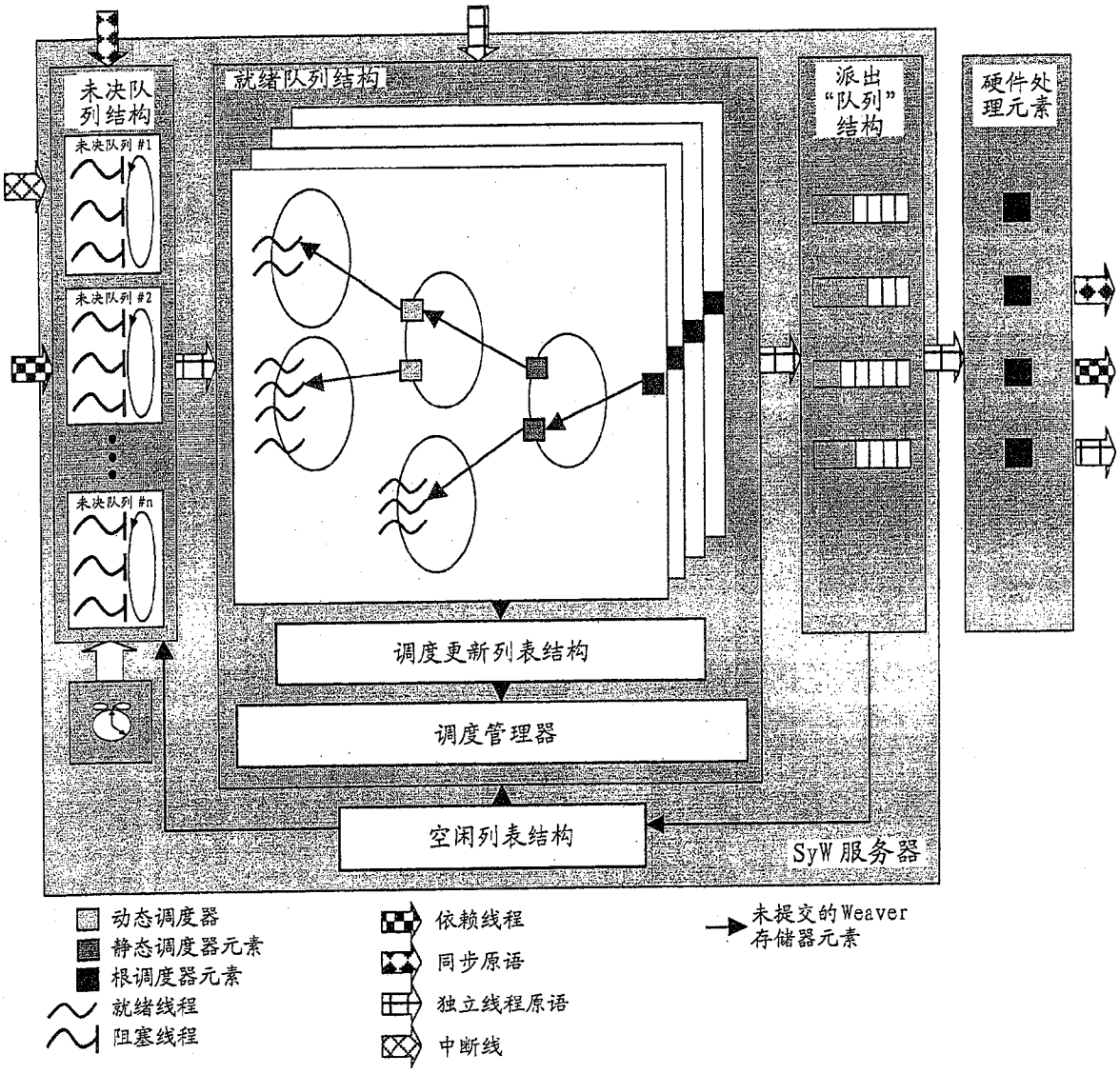


图 39 :SystemWeaver 服务器概念图

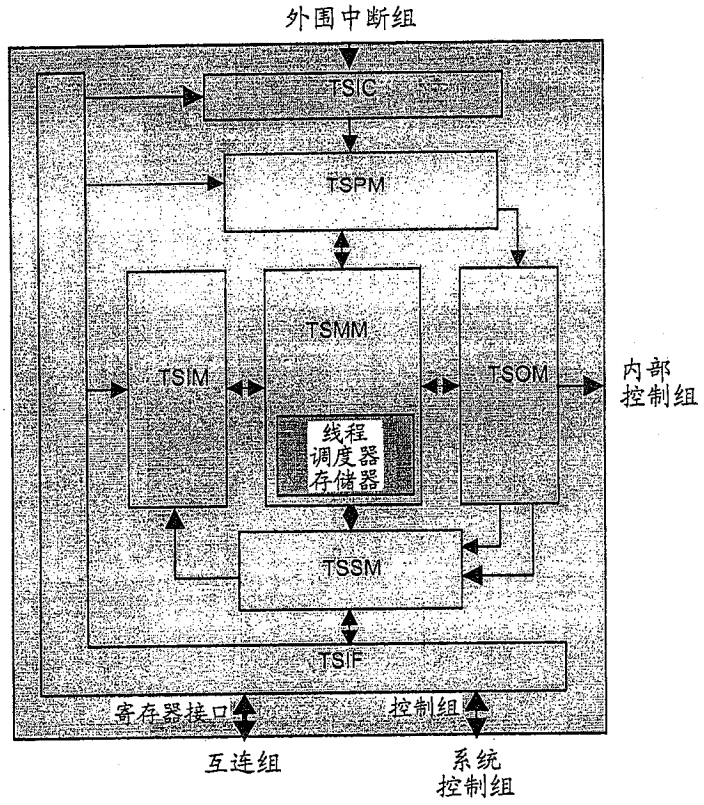

外围中断组
TSIC
TSPM
TSMM
TSIM
TSOM
内部
控制组
线程
调度器
存储器
TSSM
TSIF
寄存器接口
控制组
互连组
系统
控制组

图 40

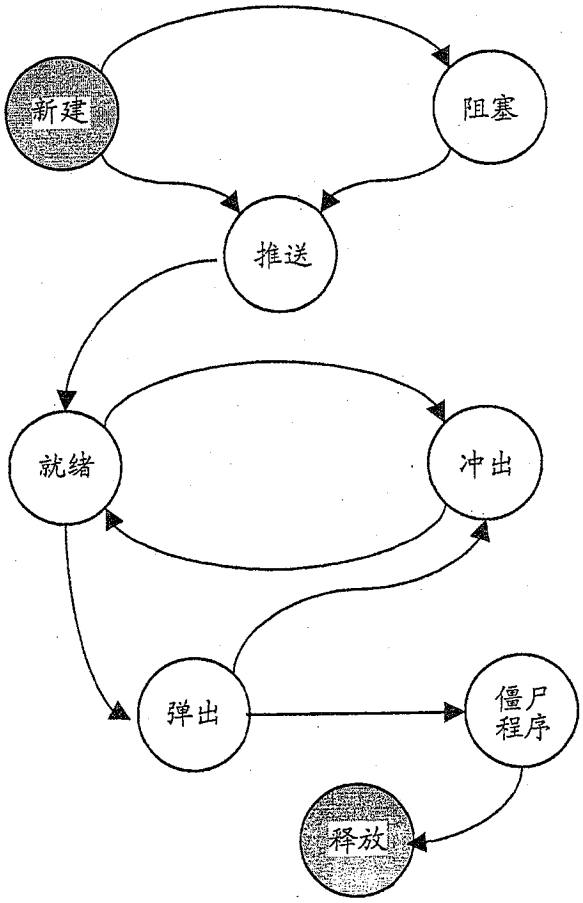


图 41 :SystemWeaver 内部线程状态图

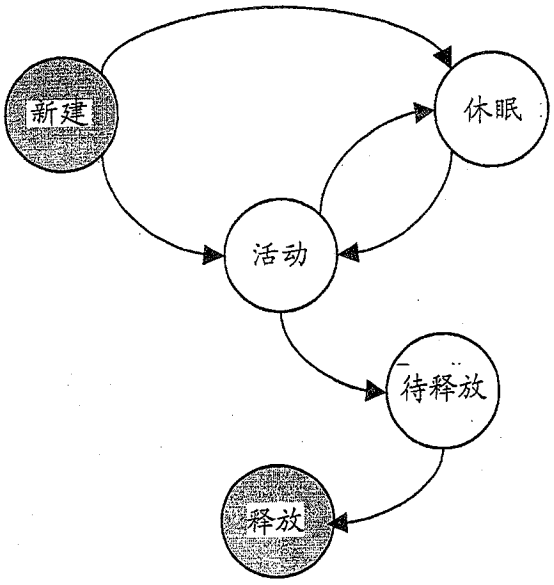


图 42 :SystemWeaver 调度器层状态图

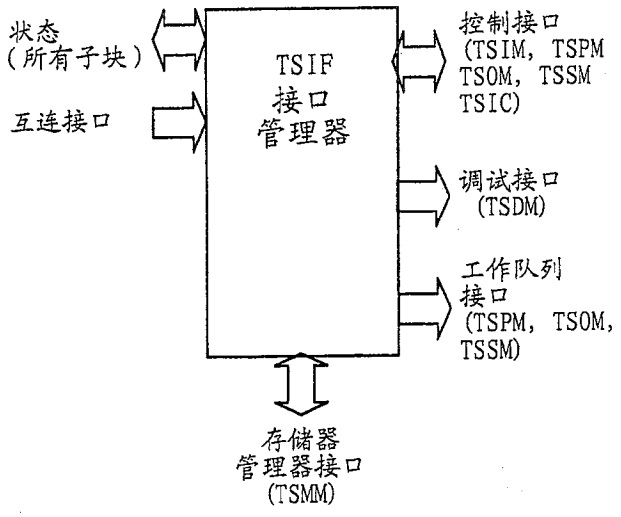


图 43 :TSIF 的主要 IO

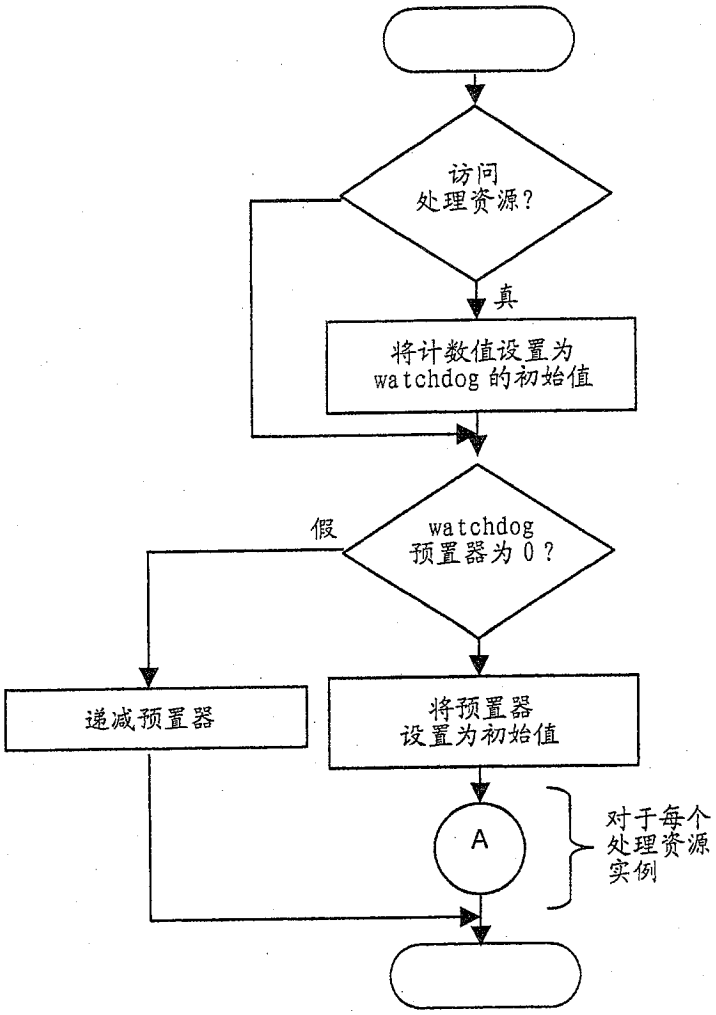


图 44 :每个循环的 Watchdog 中断控制

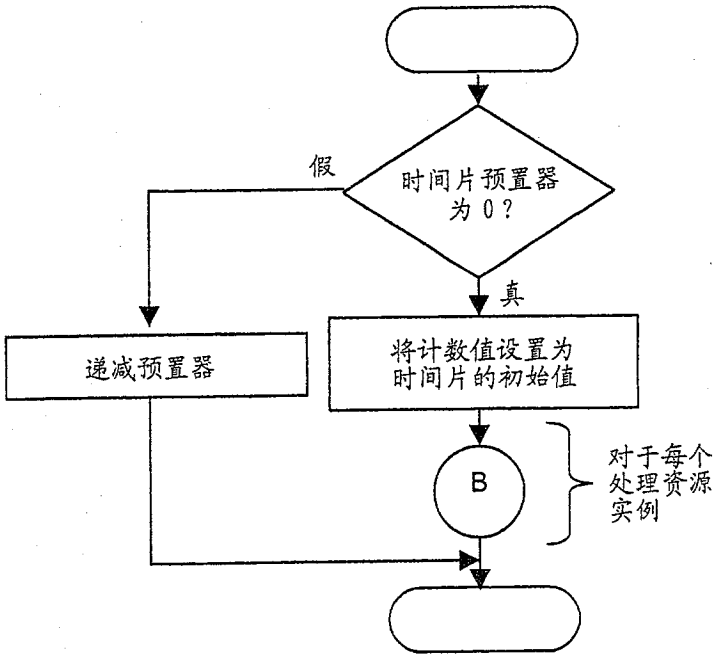


图 45 :时间片控制

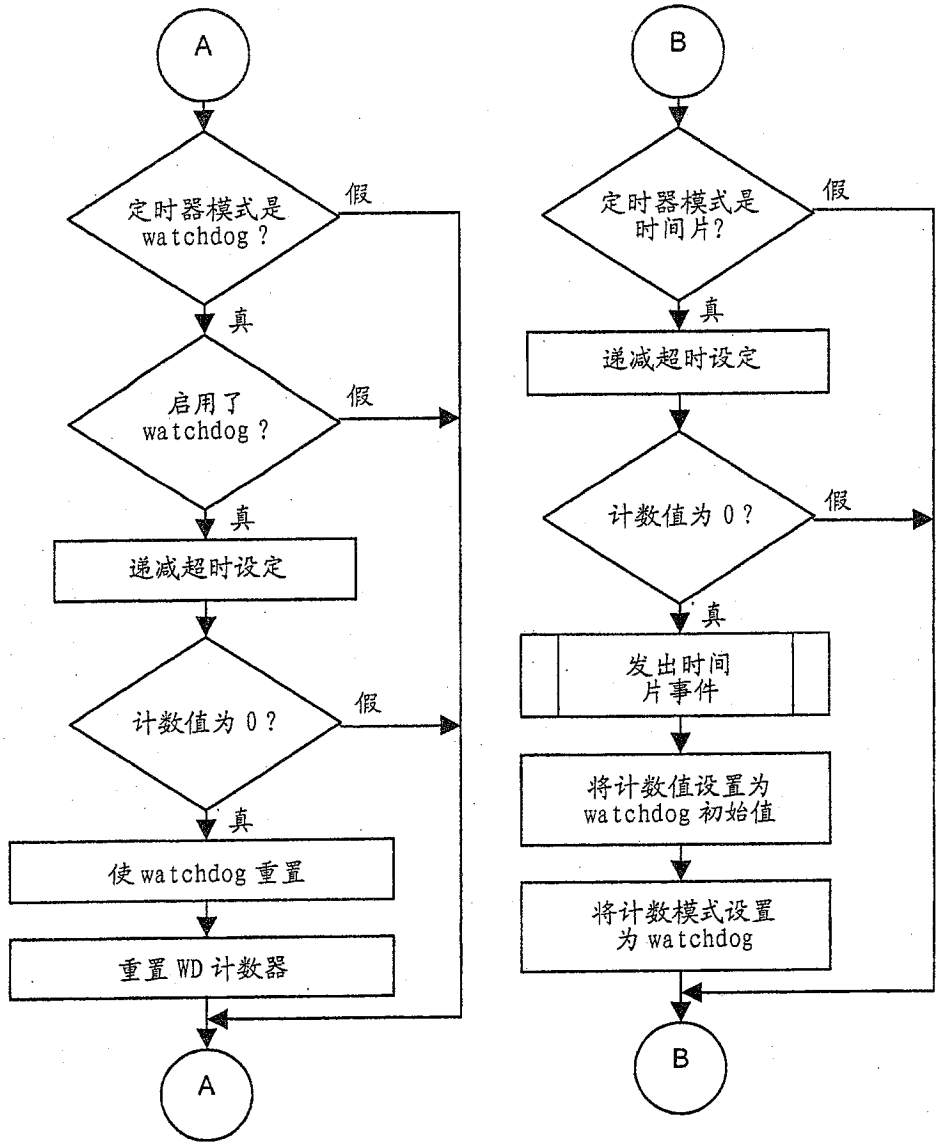


图 46 :活动定时器循环流程图

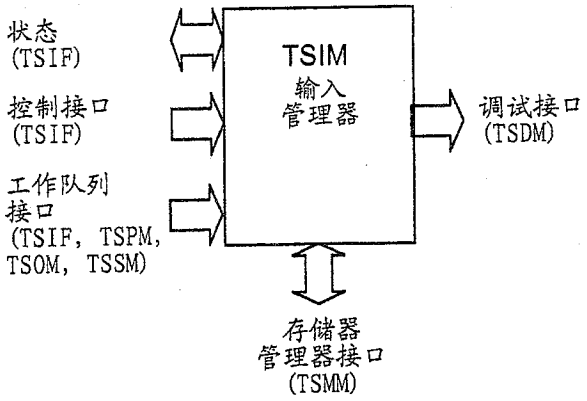


图 47 :TSIM 的主要 IO

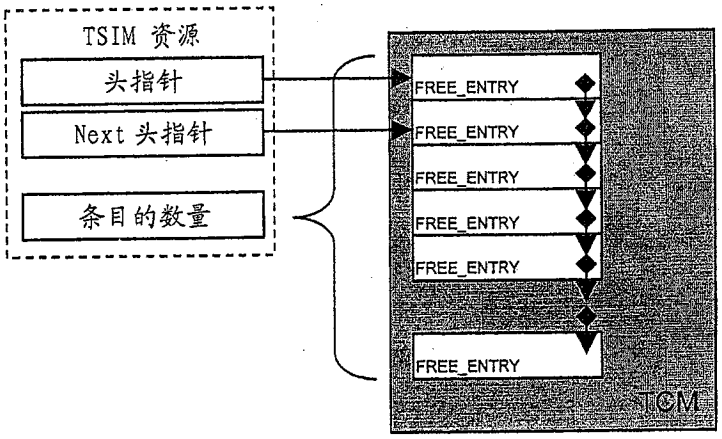


图 48 :TSIM 内部架构

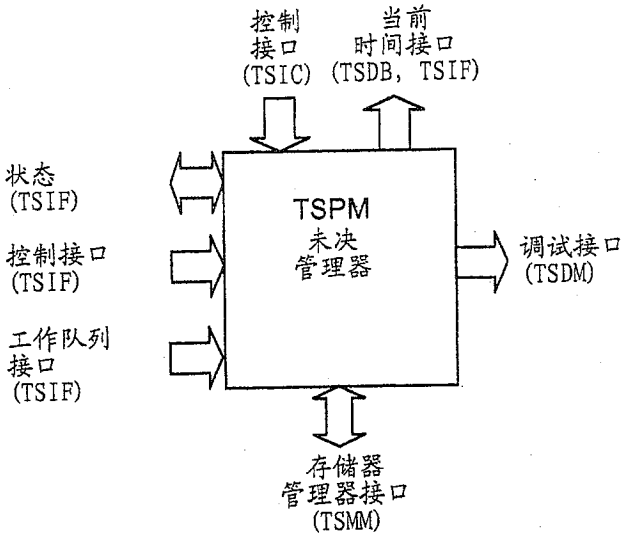


图 49 :TSPM 的主要 IO

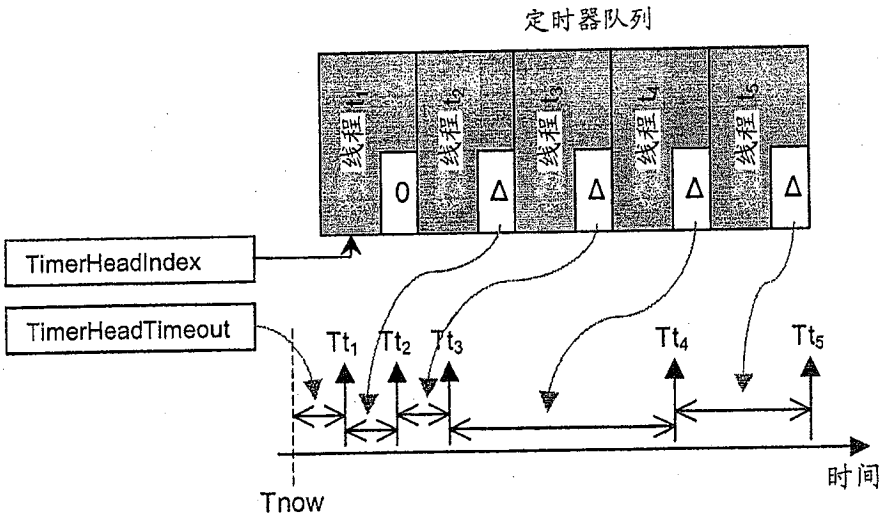


图 51 :基本的定时器队列 结构（等待状态）

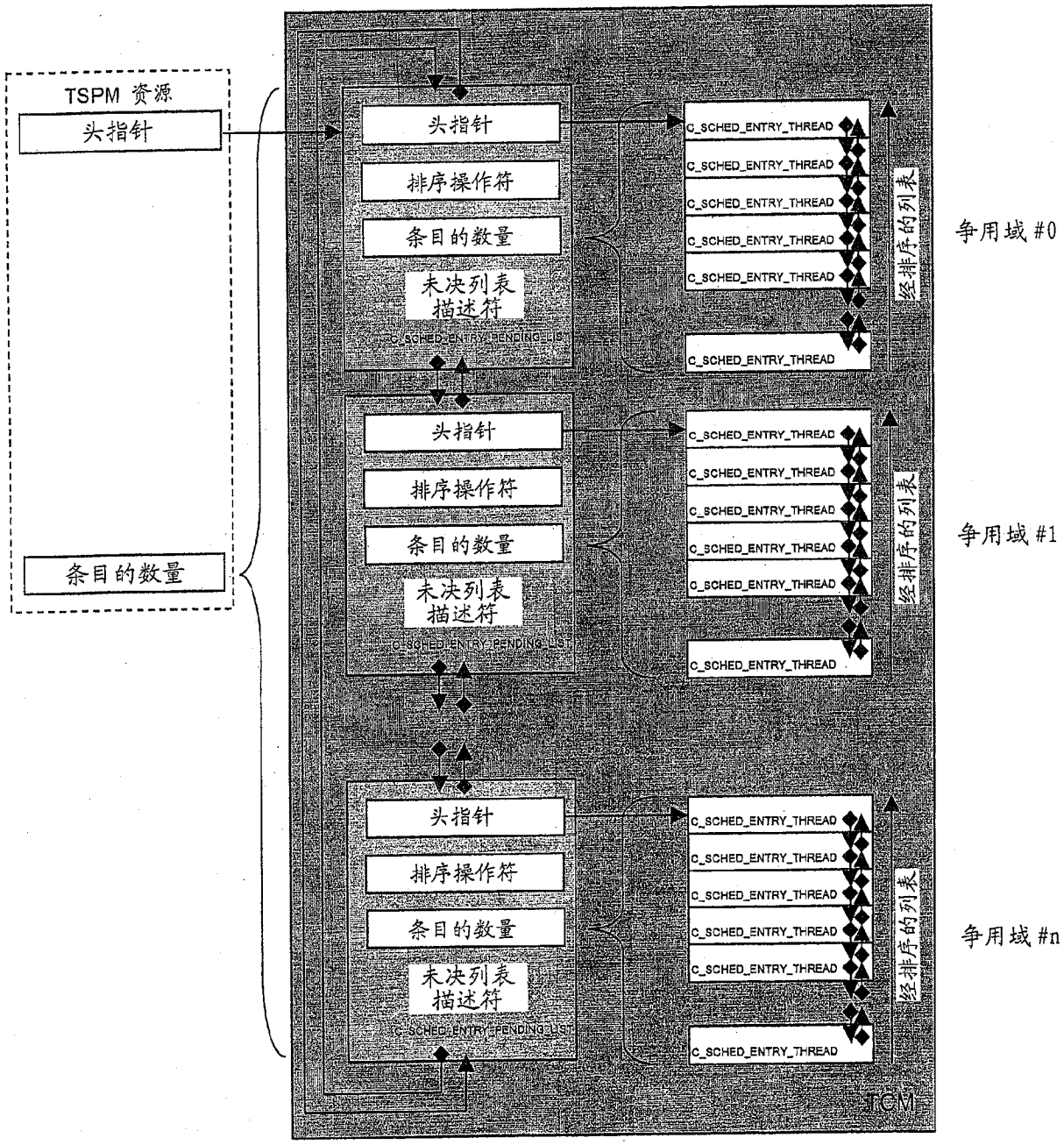


图 50 :TSPM 未决队列结构

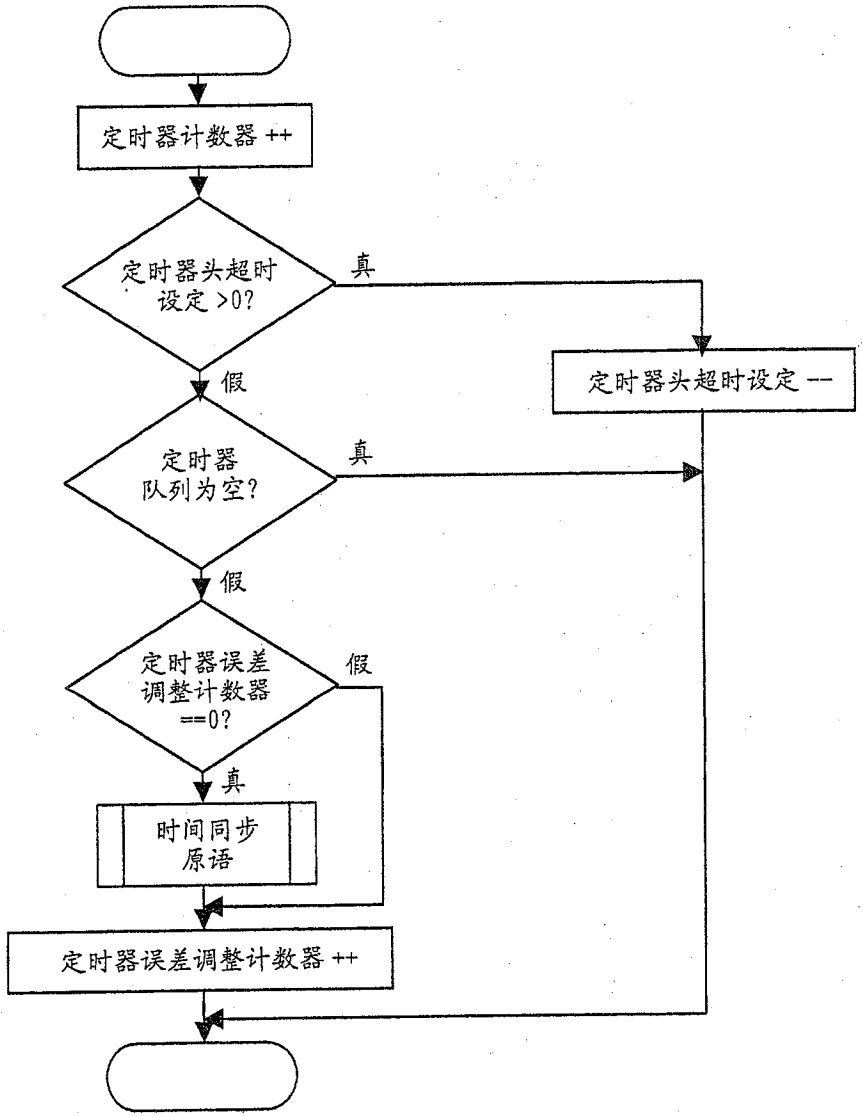


图 52 :活动定时器循环流程图

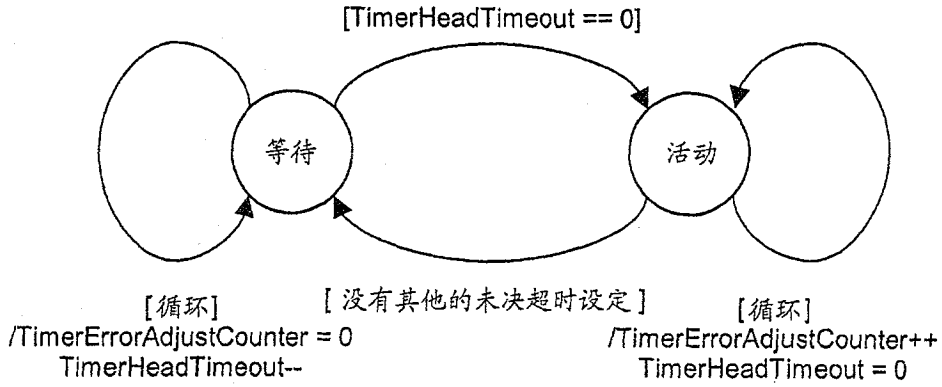


图 53 :基本超时状态机

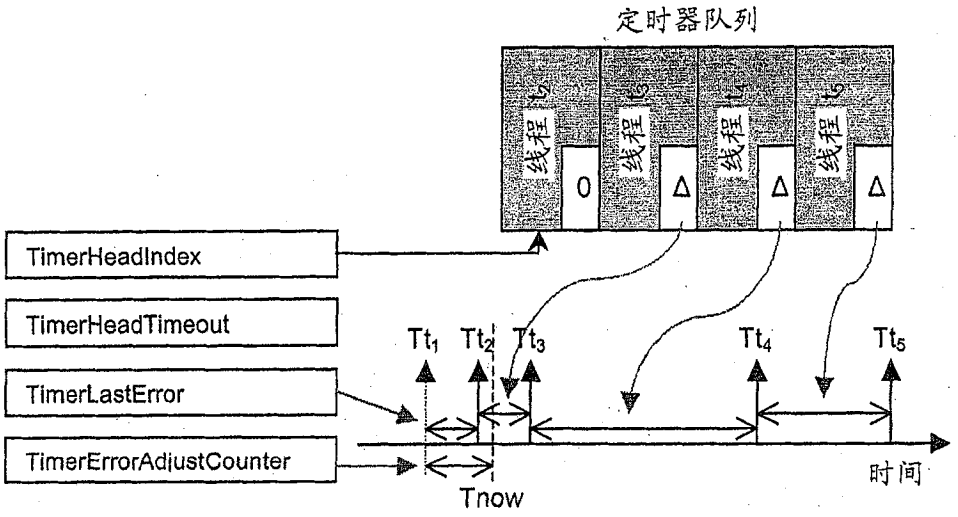


图 54 :弹出线程 1 之后的基本 定时器队列结构

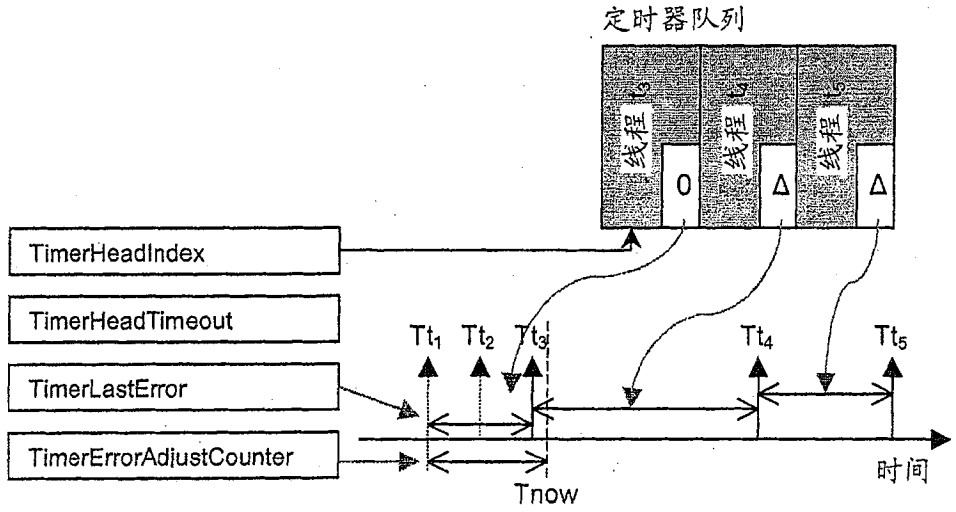


图 55 :弹出线程 2 之后的基本 定时器队列结构

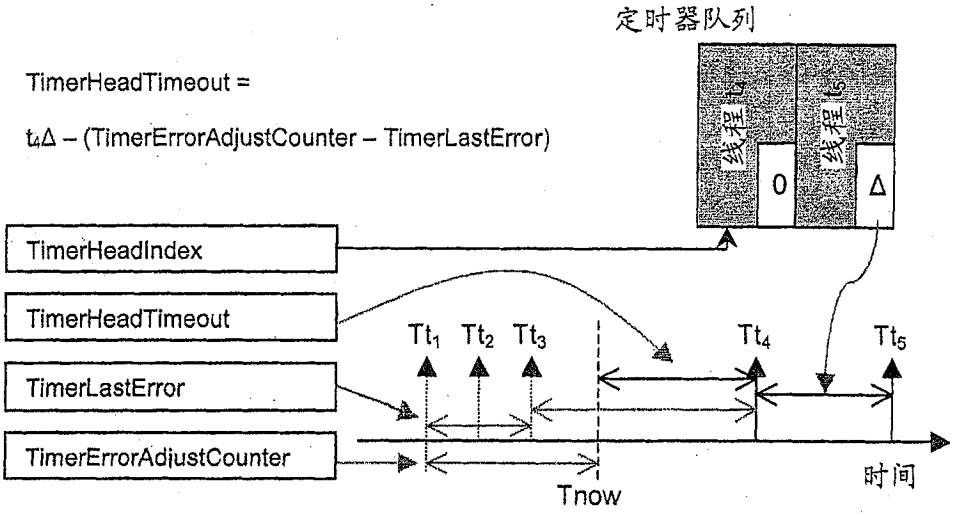


图 56 :弹出线程 3 之后的基本 定时器队列结构

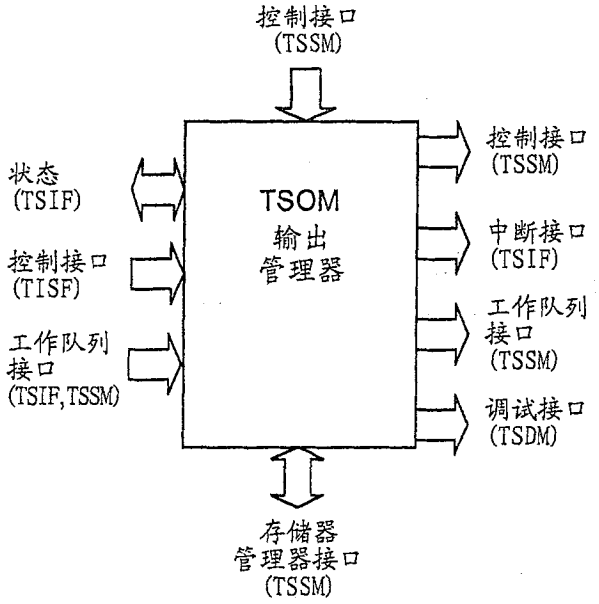


图 57 :TSOM 的主要 IO

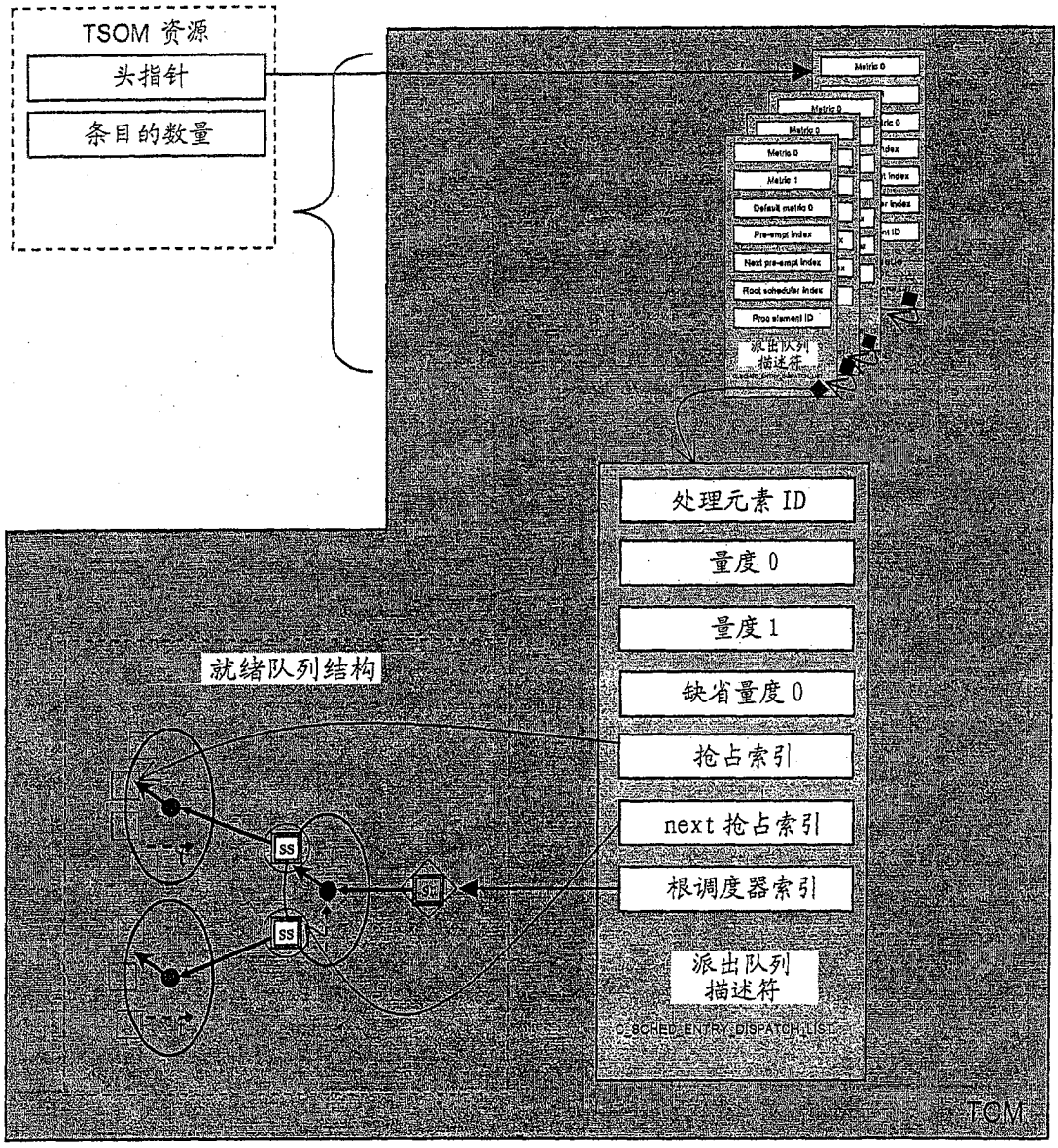


图 58 :TSOM 内部架构

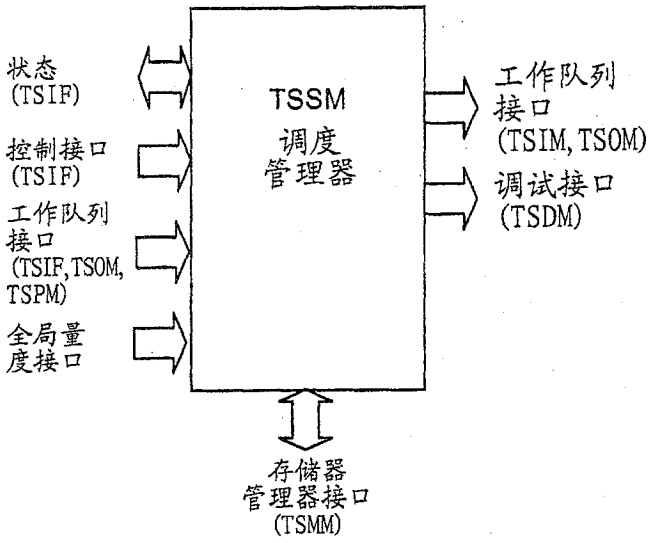


图 59 :TSSM 的主要 IO

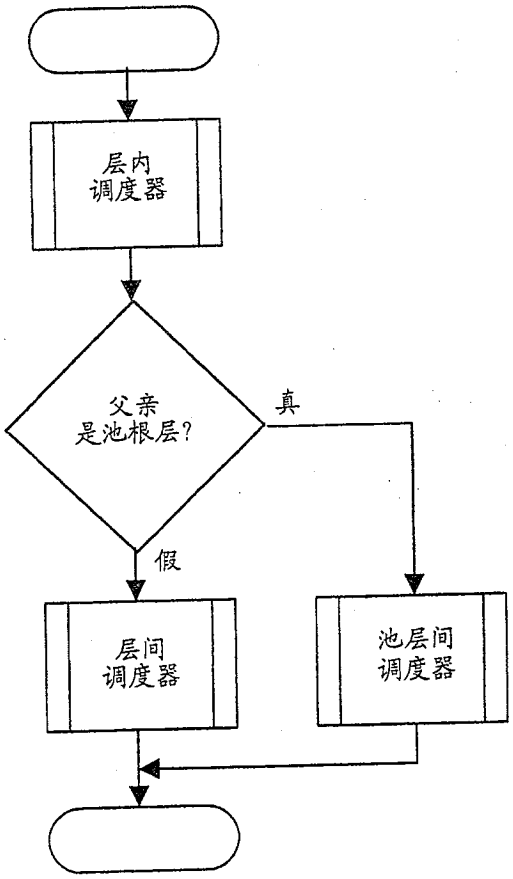


图 60 :重新调度的流程

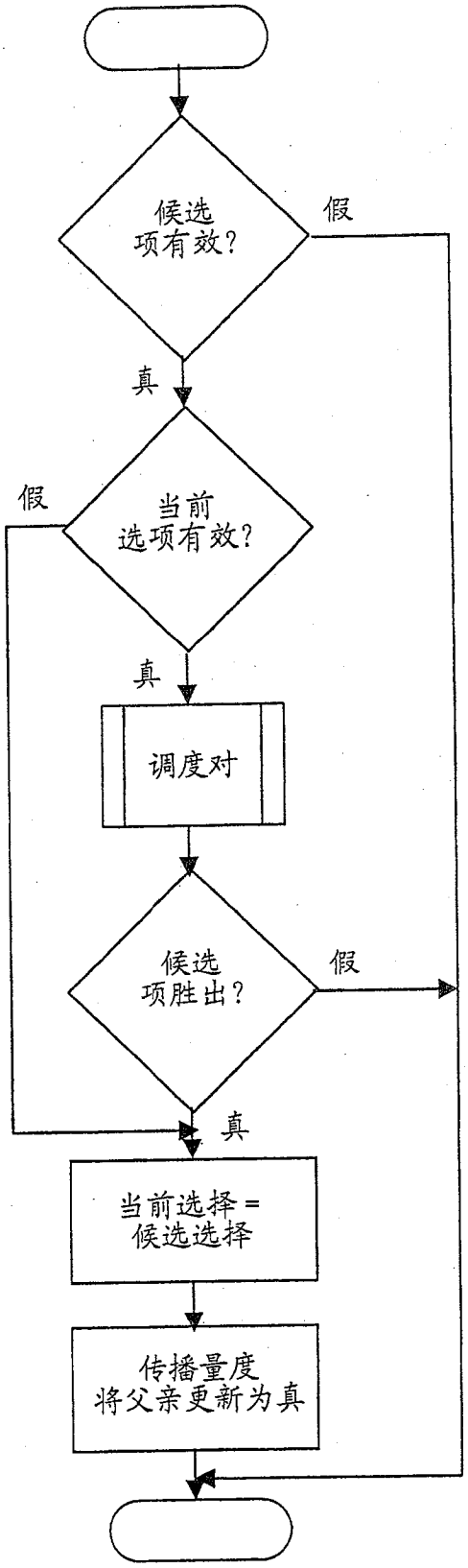


图 61 :单次调度操作

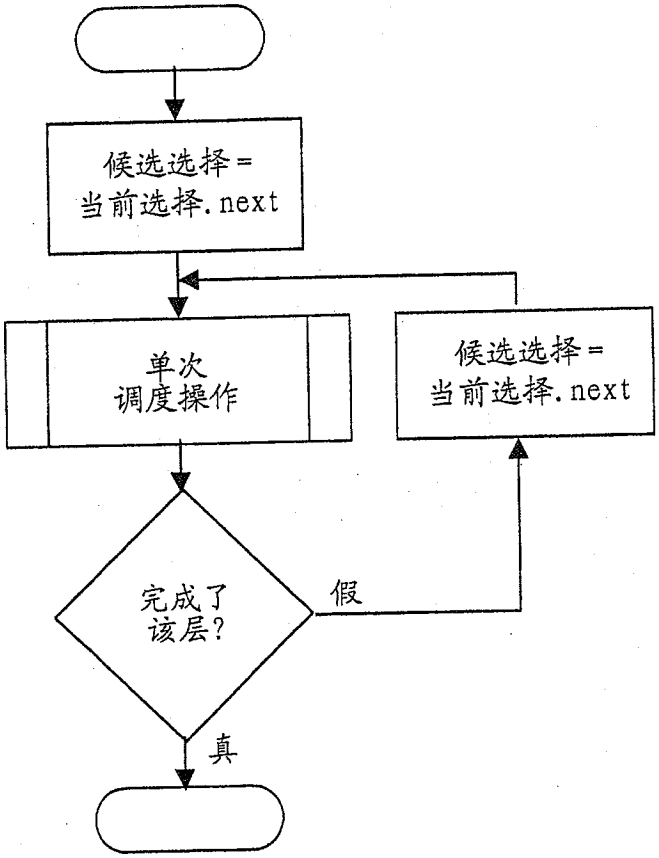


图 62 :弹出和一般调度 操作的层内调度

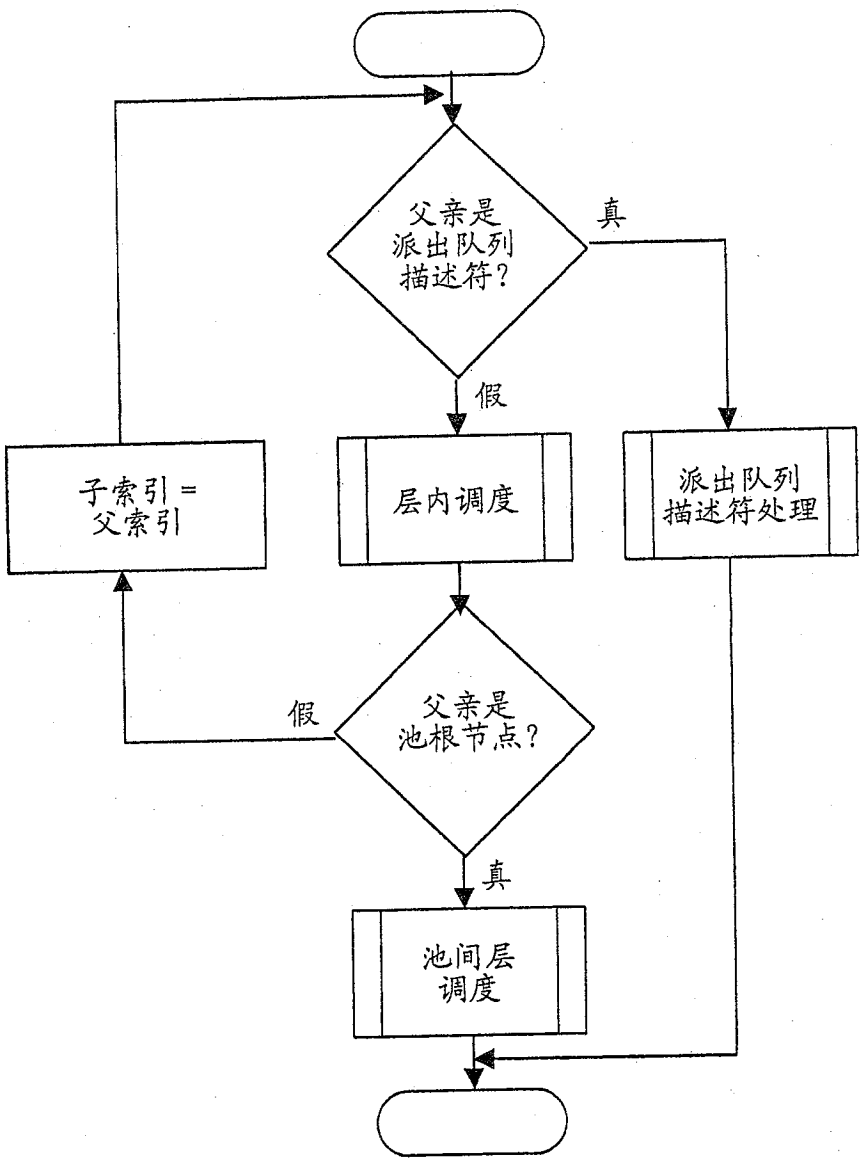


图 63 :层间调度器处理

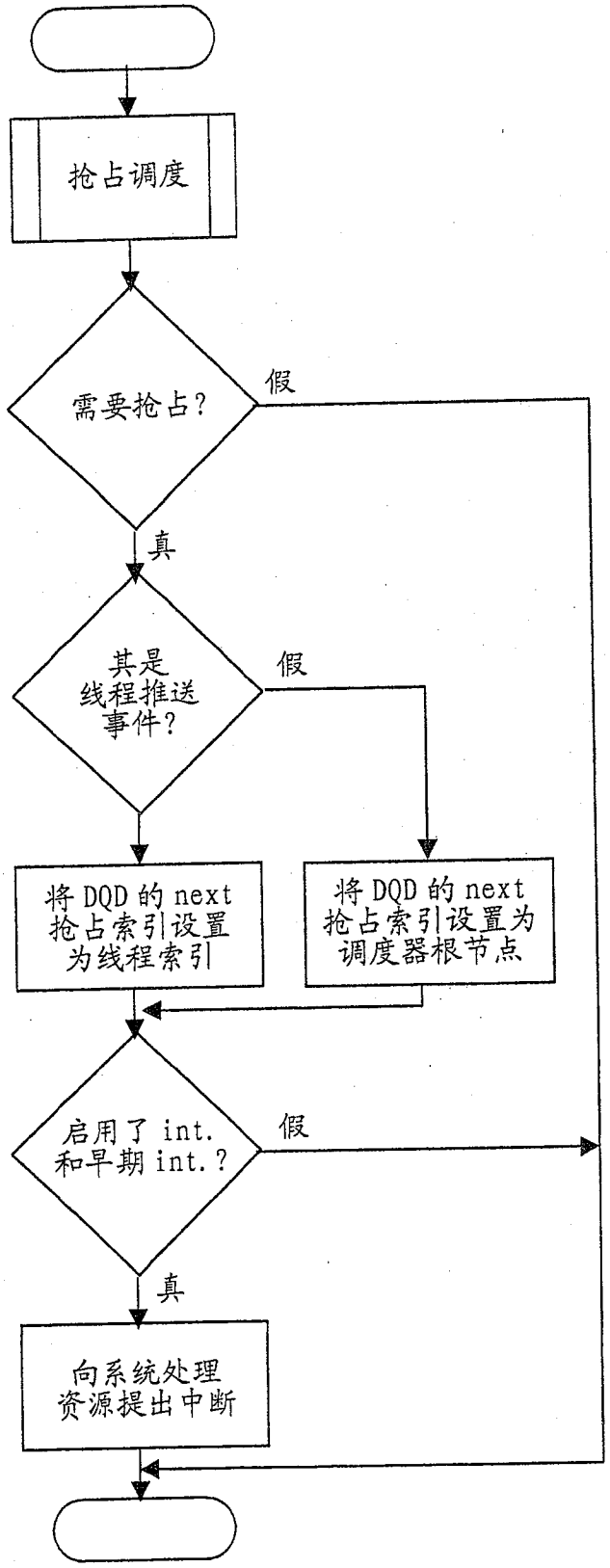


图 64 :层间调度器 派出队列处理

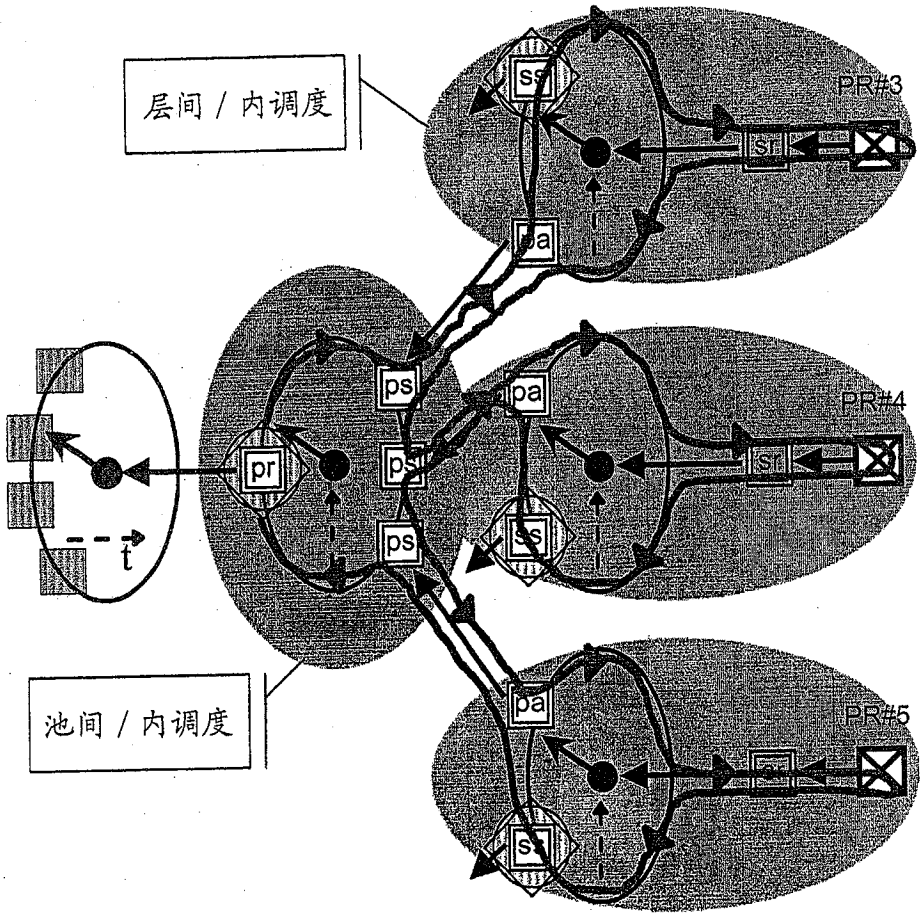


图 65 :池间调度中的迭代池分配层

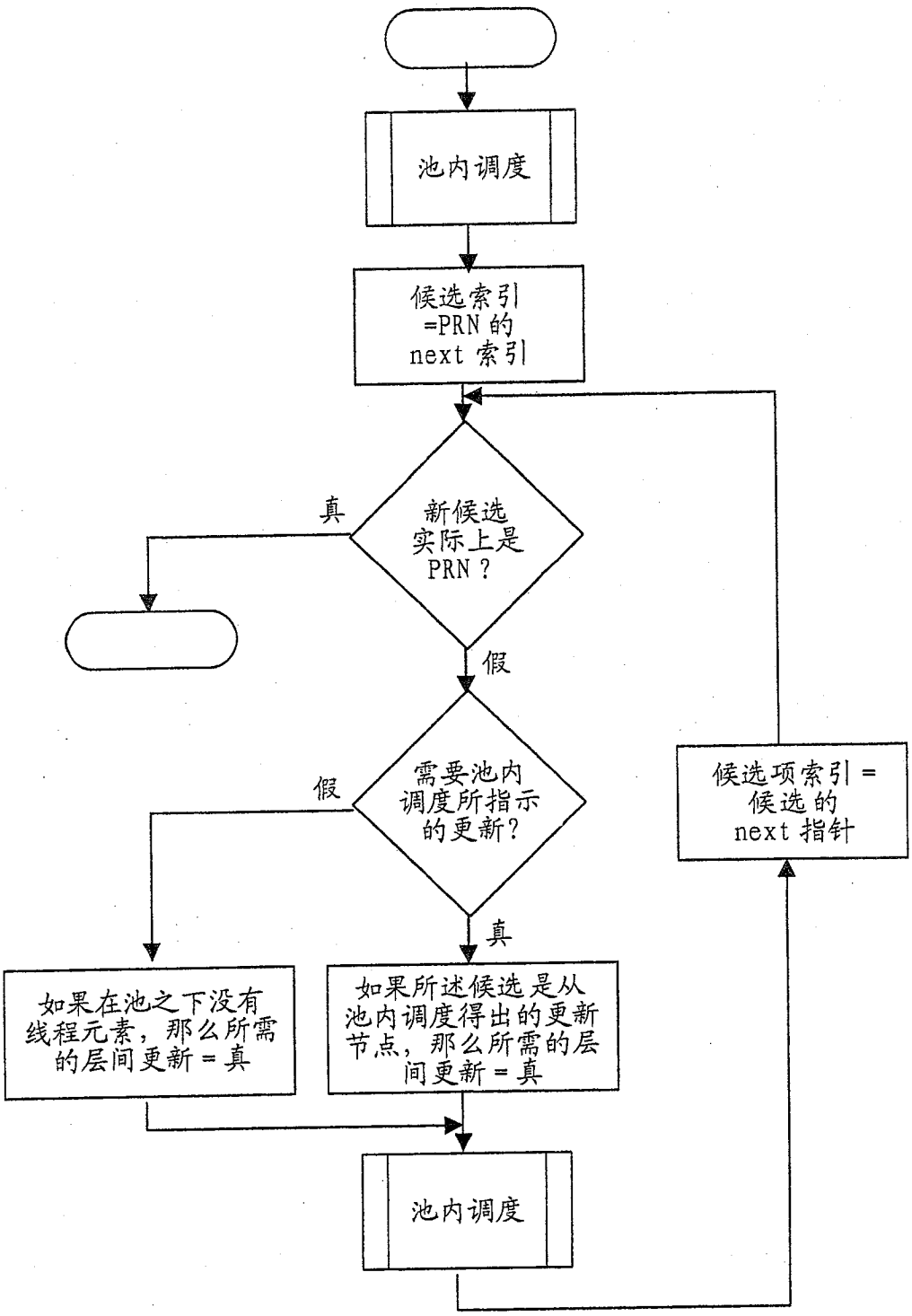


图 66 :池间分配层 调度器处理

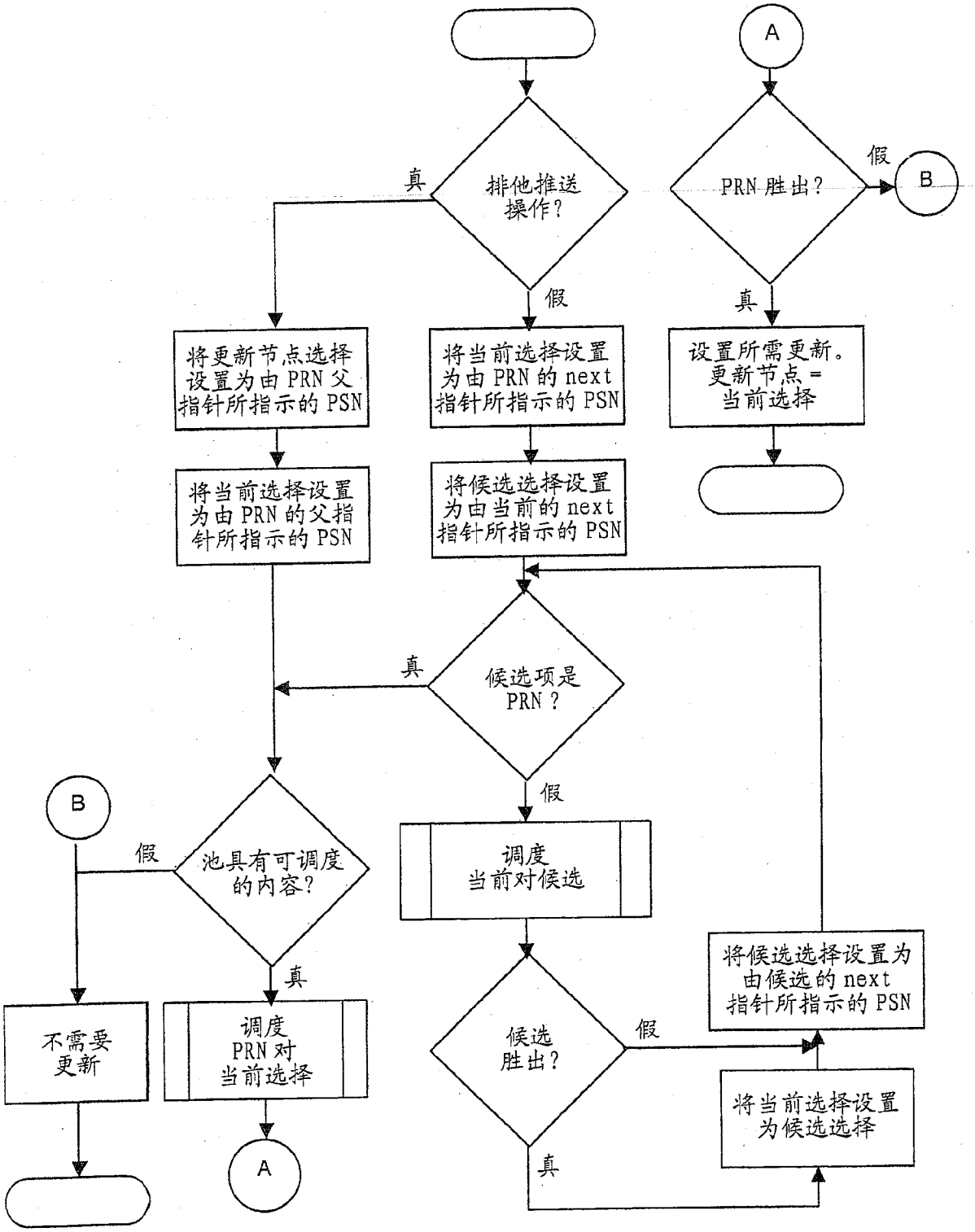


图 67 :池内层调度器处理

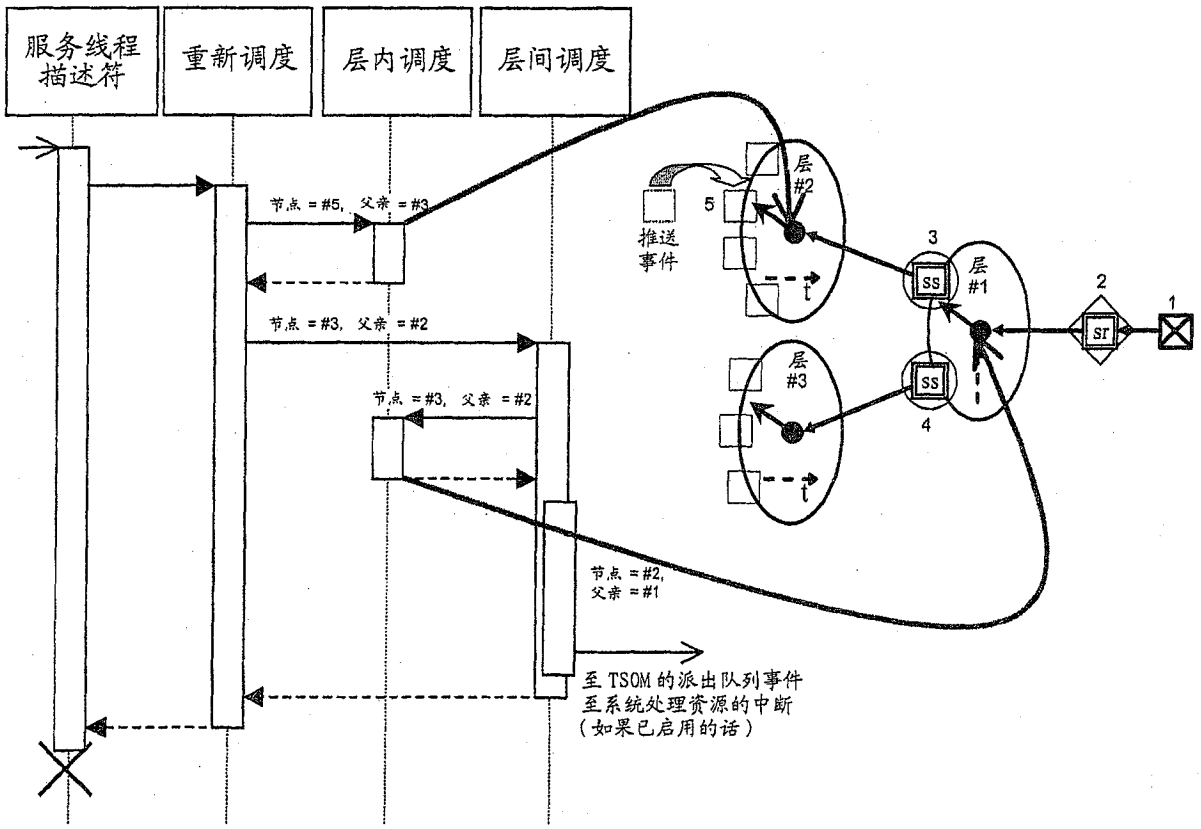


图 68 :基本调度层级序列图

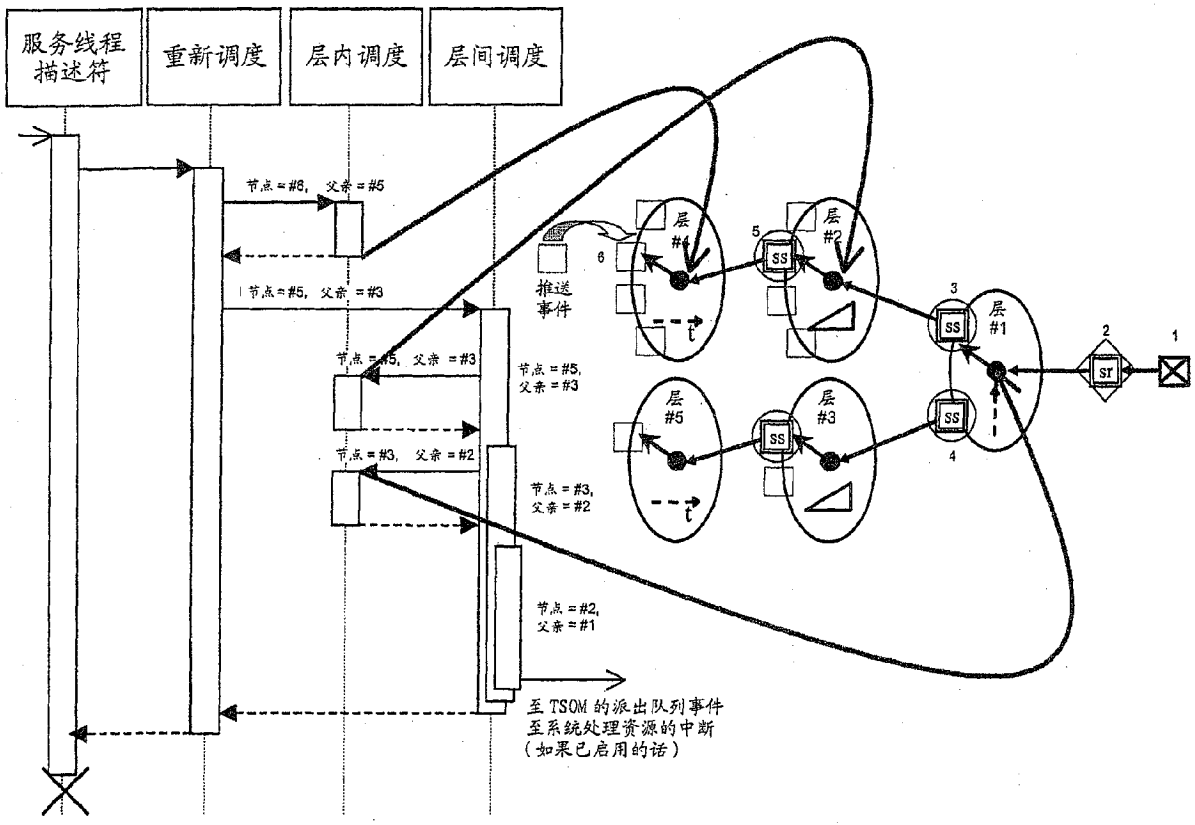


图 69 :级联调度层级序列图

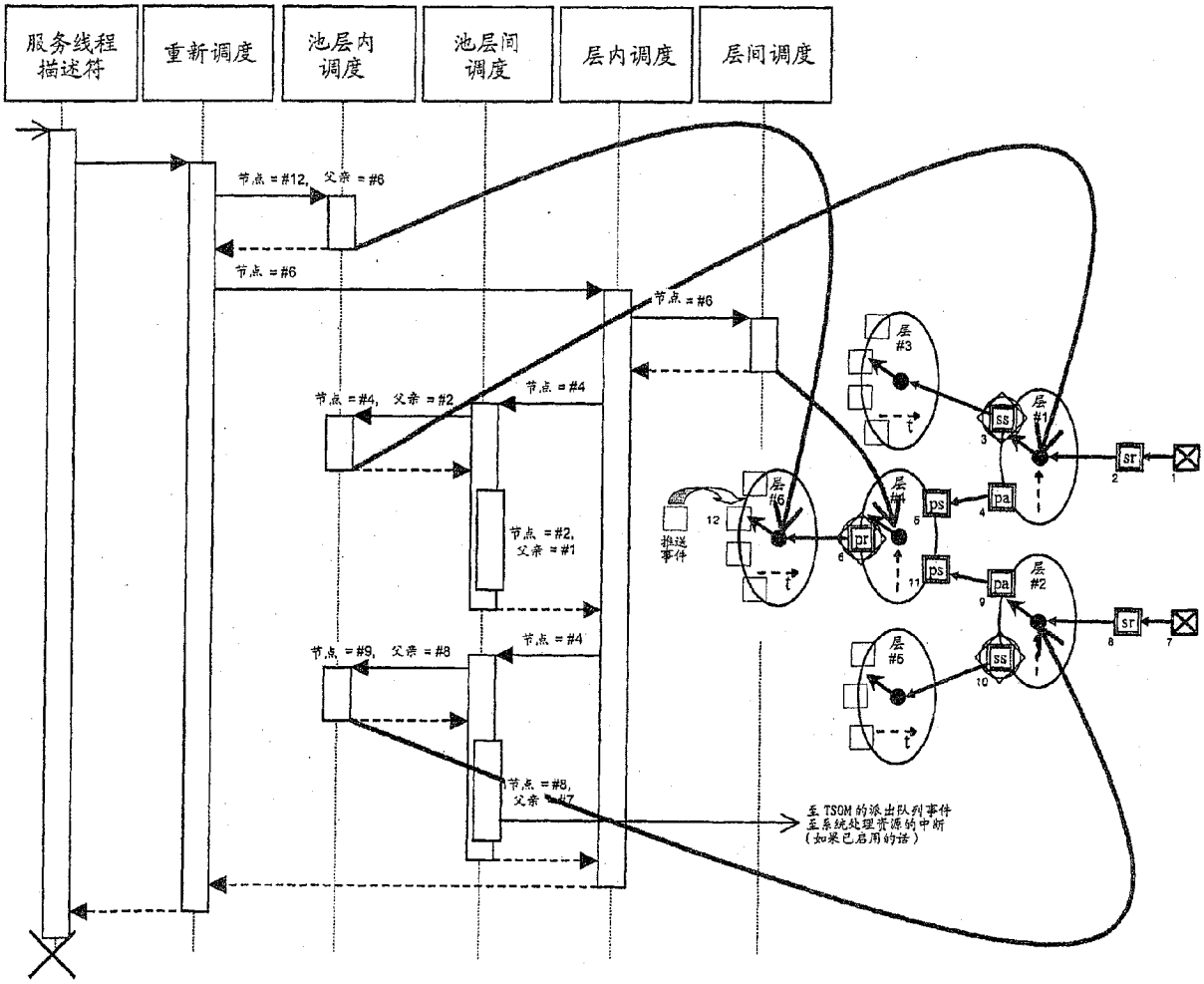


图 70 :基于池的调度层级序列图

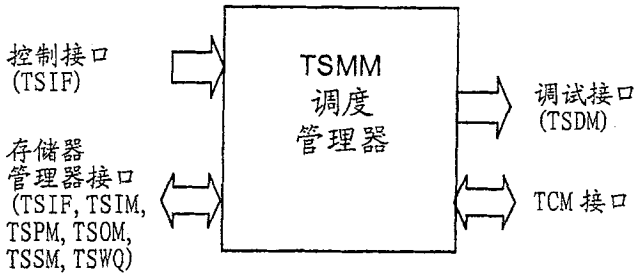


图 71 :TSMM 的主要 IO

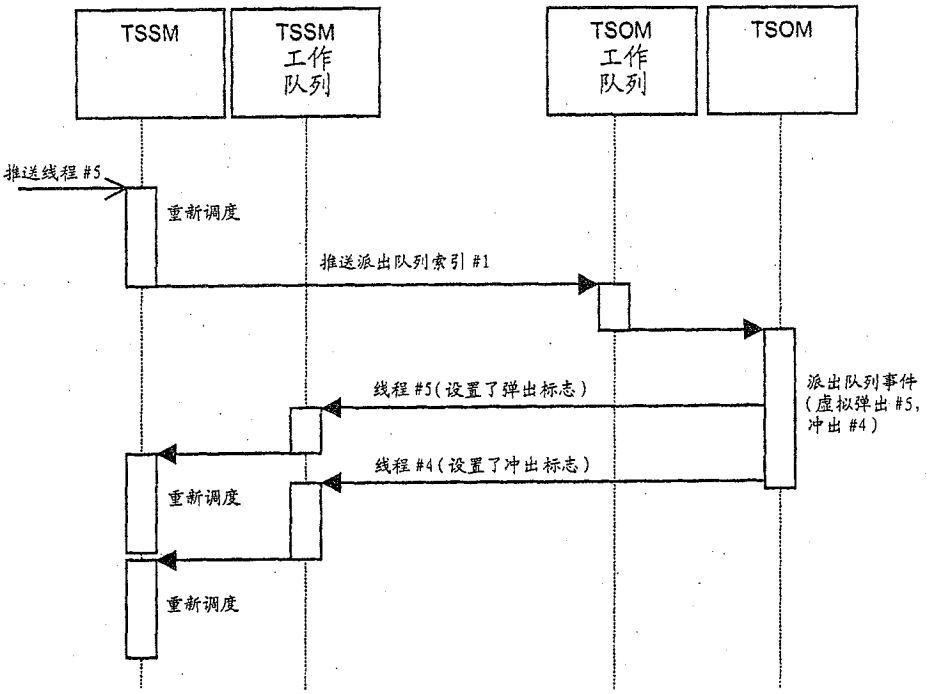


图 72 :推送事件 TSOM、TSSM 交互

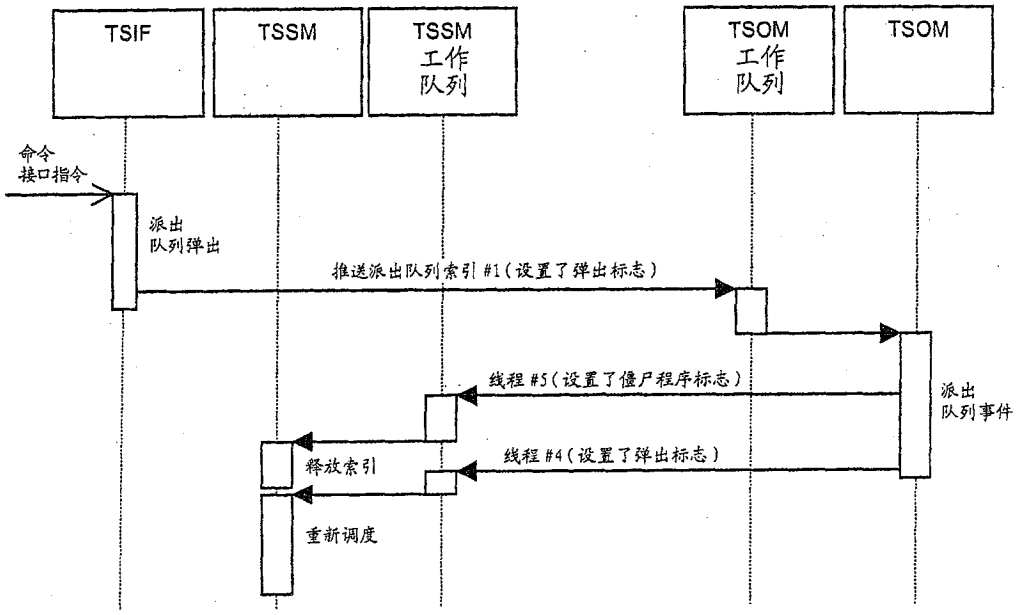


图 73 :弹出事件 TSOM、TSSM 交互

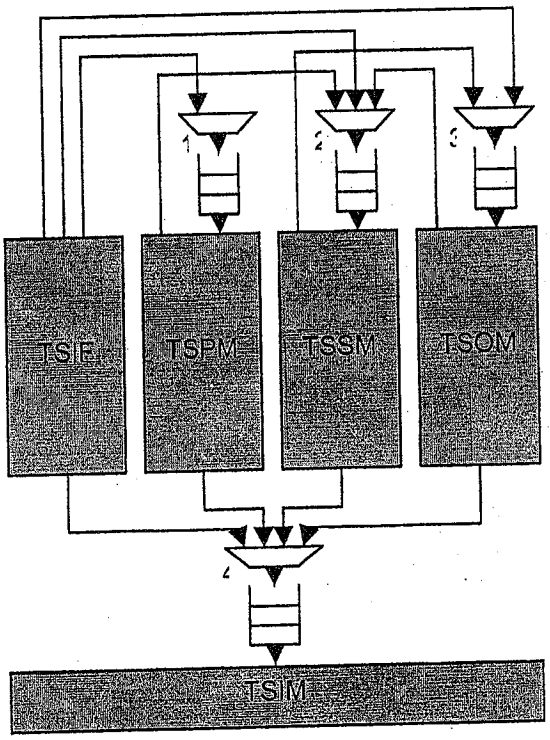


图 74 :SystemWeaver 服务器 子块间工作队列架构