

[19] 中华人民共和国国家知识产权局



[12] 发明专利说明书

专利号 ZL 200710084738.0

[51] Int. Cl.

G06F 9/46 (2006.01)

H04N 7/24 (2006.01)

H04N 7/26 (2006.01)

[45] 授权公告日 2009 年 10 月 14 日

[11] 授权公告号 CN 100549963C

[22] 申请日 2007.2.28

[21] 申请号 200710084738.0

[30] 优先权

[32] 2006.5.4 [33] US [31] 11/417,838

[73] 专利权人 国际商业机器公司

地址 美国纽约

[72] 发明人 顾晓晖 王海勋 P·S-L·渝

[56] 参考文献

US2005/0081116A1 2005.4.14

US6728753B1 2004.4.27

CN1729677A 2006.2.1

US2005/0169255A1 2005.8.4

审查员 孙 芳

[74] 专利代理机构 北京市中咨律师事务所

代理人 于 静 李 峰

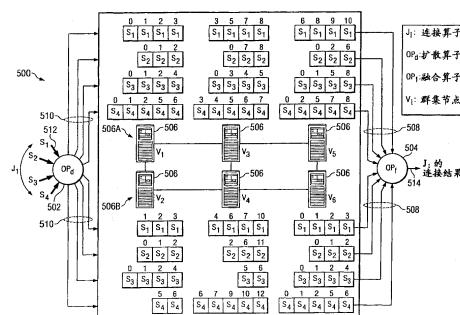
权利要求书 3 页 说明书 29 页 附图 12 页

[54] 发明名称

用于多路数据流相关的可扩缩处理的系统和方法

[57] 摘要

一种用于处理多路流相关的计算机实现的方法、装置以及计算机可用程序代码。接收流数据用于相关。形成用于连续地将多路流相关工作负荷划分成较小的工作负荷片的任务。所述较小的工作负荷片中的每一个均可以由单个主机处理。将所述流数据发送至不同的主机用于相关处理。



1. 一种用于处理多路流相关的计算机实现的方法，所述计算机实现的方法包括：

接收流数据用于进行多路流之间的相关处理，以便保持连接语义；

形成用于连续地将多路流相关工作负荷划分成较小的工作负荷片的任务，其中所述较小的工作负荷片中的每一个均可以由单个主机处理；以及

将所述流数据发送至不同的主机用于所述相关处理，

其中所述任务包括进行流划分，以便将输入流分成分段，并将所述分段分布到所述不同的主机上用于相关处理。

2. 根据权利要求 1 的计算机实现的方法，其中所述任务进一步包括进行算子划分，以便将多路相关算子分成多个较小的算子，并在不同的分布式主机上执行不同的较小的算子。

3. 根据权利要求 1 的计算机实现的方法，其中通过多个输入流上的滑动窗连接实现所述多路流相关。

4. 根据权利要求 1 的计算机实现的方法，其中所述形成步骤用于形成连续优化问题，并且其包括：

标识相关约束；

计算每个流数据的路由路径，用于确保满足所述相关约束、平衡所述不同主机的工作负荷，以及最小化所述流数据的复制开销。

5. 根据权利要求 1 的计算机实现的方法，其中所述将输入流分成分段的步骤进一步包括：

选择具有最高速率的输入流作为主流，并且将其它的流指示为从流；

将所述主流划分成不相交的分段；以及

将所述从流划分成重叠的分段以符合相关约束。

6. 根据权利要求 5 的计算机实现的方法，其中所述选择步骤在运行

时是动态改变的。

7. 根据权利要求 6 的计算机实现的方法，其中所述选择步骤的动态改变包括：

响应于所述主流慢于所述其它的流中的任何一个，触发流角色交换，用于选择所述任何一个具有最高速率的输入流作为主流，并且将其它的流指示为从流；以及

根据所述相关约束采用转换阶段来实现所述流角色交换。

8. 根据权利要求 5 的计算机实现的方法，其中所述将所述主流划分成不相交的分段的步骤用于调整所述不相交的分段的分段长度，并且其进一步包括：

当系统条件改变时触发分段自适应；

进行基于采样的仿形过程，以便搜索新的分段长度。

9. 根据权利要求 2 的计算机实现的方法，其中所述将多路相关算子分成多个较小的算子的步骤进一步包括：

确定连接顺序，以便将多路相关处理分成多跳；

在每跳查找相关分段的位置；以及

在每跳计算覆盖所有所述相关分段的最小主机集合。

10. 根据权利要求 9 的计算机实现的方法，其中由在不同流之间所观察到的连接选择性来确定所述连接顺序。

11. 根据权利要求 9 的计算机实现的方法，其中使用路由表进行所述查找步骤，其中所述路由表包括不同输入流的滑动窗中相关分段的布局信息。

12. 根据权利要求 9 的计算机实现的方法，其中为了最小化中间结果传输，所述在每跳计算覆盖所有所述相关分段的最小主机集合的步骤进一步包括：

考虑前一跳的主机集合选择；以及

重新使用由所述前一跳选择的主机。

13. 一种用于处理多路流相关的系统，其包括：

数据处理系统，所述数据处理系统用于接收来自于输入流的流数据进行多路流之间的相关处理以便保持连接语义，以及用于连续地将多路流相关工作负荷划分成较小的工作负荷片的任务形成，其中所述较小的工作负荷片中的每一个均可以由单个主机处理，其中所述任务用于进行算子划分，以便将多路相关算子分成多个较小的算子、在分布式主机上执行不同的较小的算子，以及用于进行流划分，以便将输入流分成分段，并且将所述分段分布到主机上进行相关处理；以及

多个主机，所述多个主机可操作连接至所述数据处理系统；

其中所述数据处理系统将所述流数据发送至所述多个主机，用于所述相关处理。

14. 根据权利要求 13 的系统，其中所述数据处理系统选择具有最高速率的输入流作为主流，并且将其它的流指示为从流，将所述主流划分成不相交的分段，并且将所述从流划分成重叠的分段以符合相关约束。

15. 一种用于处理多路流相关的计算机系统，所述计算机系统包括：

接收装置，所述接收装置用于接收流数据用于进行多路流之间的相关处理，以便保持连接语义；

形成装置，所述形成装置用于形成连续地将多路流相关工作负荷划分成较小的工作负荷片的任务，其中所述较小的工作负荷片中的每一个均可以由单个主机处理；以及

发送装置，所述发送装置用于将所述流数据发送至不同的主机用于所述相关处理，

其中所述任务包括进行流划分，以便将输入流分成分段，并将所述分段分布到所述不同的主机上用于相关处理。

用于多路数据流相关的可扩缩处理的系统和方法

技术领域

本发明一般涉及一种改进的数据处理系统，并且特别地，涉及一种用于处理数据流的计算机实现的方法和装置。更特别地，本发明涉及一种用于多路数据流相关的可扩缩处理的计算机实现的方法、装置以及计算机可用程序代码。

背景技术

流处理计算应用是满足关于数据的一些限制的应用，在该应用中数据以信息流的形式进入系统。注意到正在处理的数据的容量可能太大而无法存储；因此，信息流要求对动态数据流进行复杂的实时处理，例如传感器数据分析以及网络业务监控。流处理计算应用的例子包括视频处理、音频处理、流式数据库以及传感器网络。在这些应用中，来自外部源的数据流流入数据流管理系统，其在此由不同的连续查询算子处理。

为了支持无界流，流处理系统将滑动窗与每个流相关联。滑动窗含有在流上最近到达的数据项。窗口可以是基于时间的，例如在最后 60 秒中到达的视频帧，或者是基于数字的，例如最后 1000 个视频帧。最重要的连续查询算子之一是在多个不同的数据流上的滑动窗连接。滑动窗连接的输出含有满足预先定义的连接判定（join predicate）的、并且在其相应的窗口中被同时显现的相关元组的所有集合。

一些实例应用包括在不同的新闻视频流中搜索类似的图像用于热门话题检测，以及在不同的网络业务流中对源/目的地址进行相关用于入侵检测。基于关键字的同等连接（equijoin）可能不太有效，因为很多流相关应用要求比关键字比较更复杂的连接判定。例如，在新闻视频相关应用中，连接条件是两个图像的 40 维分类值之间的距离是否低于阈值。因而，不同

流的相关数据意味着在不同流上找到满足一个或多个预先定义的相关判定的那些数据。

处理多路流连接的主要挑战在于在多个大容量并且时变的数据流上实时地进行大量的连接比较。给定高流速和大窗口尺寸，加窗的流连接可能会有大的存储需求。此外，诸如图像比较的一些查询处理还可以是中央处理器密集的。单个主机可能由于多路流连接工作负荷而易于过载。

发明内容

本说明性实施例提供了一种用于处理多路流相关的计算机实现的方法、装置以及计算机可用程序代码。接收流数据用于相关。形成用于连续地将多路流相关工作负荷划分成较小的工作负荷片的任务。所述较小的工作负荷片中的每一个均可以由单个主机来处理。将所述流数据发送至不同的主机用于相关处理。

附图说明

在所附权利要求中阐述了被认为是说明性实施例的特性的新颖的特征。然而，当结合附图阅读时，通过参照以下对说明性实施例的具体描述，将最好地理解说明性实施例本身，以及使用的优选模式及其更多的目的和优势，其中：

图 1 是可以在其中实现说明性实施例的数据处理系统的网络的图形表示；

图 2 是可以在其中实现说明性实施例的数据处理系统的框图；

图 3 描述了依照说明性实施例的流处理的例子；

图 4 是依照说明性实施例的滑动窗流连接算子模型的示图；

图 5 是依照说明性实施例的分布式多路流连接执行模型的示图；

图 6A-6B 是依照说明性实施例的相关感知元组路由方案的示图；

图 7 是依照说明性实施例的对准元组路由模型的示图；

图 8 是依照说明性实施例的约束元组路由模型的示图；

-
- 图 9 是依照说明性实施例的集中式连接方法的流程图；
图 10 是依照说明性实施例的分布式连接方法的流程图；
图 11 是依照说明性实施例的对准元组路由方法的流程图；
图 12A-12B 是依照说明性实施例的约束元组路由方法的流程图；
图 13 是依照说明性实施例的多路流连接算法；
图 14 是依照说明性实施例的对准元组路由算法；以及
图 15 是依照说明性实施例的约束元组路由算法。

具体实施方式

现参照附图，并且特别参照图 1-2，提供了可以在其中实现说明性实施例的数据处理环境的示例图。应当理解图 1-2 仅是示例性的，并不旨在主张或暗示对关于可以在其中实现不同实施例的环境的任何限制。可以对所描述的环境进行很多修改。

现参照附图，图 1 描述了可以在其中实现说明性实施例的数据处理系统的网络的图形表示。网络数据处理系统 100 是可以在其中实现实施例的计算机网络。网络数据处理系统 100 含有网络 102，其是用于在网络数据处理系统 100 内连在一起的各种设备和计算机之间提供通信链路的介质。网络 102 可以包括诸如线、无线通信链路或光缆的连接。

在所描述的例子中，服务器 104 和服务器 106 与存储单元 108 一起连接至网络 102。另外，客户机 110、112 和 114 连接至网络 102。这些客户机 110、112 和 114 可以是，例如个人计算机或网络计算机。在所描述的例子中，服务器 104 向客户机 110、112 和 114 提供诸如引导文件、操作系统映像以及应用的数据。在该例中，客户机 110、112 和 114 是服务器 104 的客户机。网络数据处理系统 100 可以包括附加的服务器、客户机和未示出的其它设备。

在所描述的例子中，网络数据处理系统 100 是具有网络 102 的因特网，网络 102 代表使用传输控制协议/网际协议（TCP/IP）协议组相互通信的全世界的网络和网关的集合。处于因特网核心的是主节点或主计算机之间的

高速数据通信线路的主干线，包括路由数据和消息的数千商业的、政府的、教育的和其它的计算机系统。当然，网络数据处理系统 100 还可以实现为许多不同类型的网络，举例来说，例如内联网、局域网（LAN）或广域网（WAN）。图 1 意在举例，并不作为对不同实施例的结构限制。

现参照图 2，示出了可以在其中实现说明性实施例的数据处理系统的框图。数据处理系统 200 是诸如图 1 中的服务器 104 或客户机 110 的计算机的例子，可以在其中针对说明性实施例设置实现过程的计算机可用代码或指令。

在所描述的例子中，数据处理系统 200 使用包括北桥和存储控制器集线器（MCH）202 以及南桥和输入/输出（I/O）控制器集线器（ICH）204 的集线器体系结构。处理器 206、主存储器 208 以及图形处理器 210 耦合于北桥和存储控制器集线器 202。举例来说，图形处理器 210 可以通过加速图形端口（AGP）耦合于 MCH。

在所描述的例子中，局域网（LAN）适配器 212 耦合于南桥和 I/O 控制器集线器 204，并且音频适配器 216、键盘和鼠标适配器 220、调制解调器 222、只读存储器（ROM）224、通用串行总线（USB）端口和其它通信端口 232，以及 PCI/PCIe 设备 234 通过总线 238 耦合于南桥和 I/O 控制器集线器 204，并且硬磁盘驱动器（HDD）226 和 CD-ROM 驱动器 230 通过总线 240 耦合于南桥和 I/O 控制器集线器 204。PCI/PCIe 设备可以包括，例如以太网适配器、附加卡（add-in cards）以及用于笔记本计算机的 PC 卡。PCI 使用卡式总线控制器，而 PCIe 并不使用。ROM 224 可以是，例如闪速二进制输入/输出系统（BIOS）。硬磁盘驱动器 226 和 CD-ROM 驱动器 230 可以使用，例如集成驱动器电子电路（IDE）或串行高级技术配件（SATA）接口。超级 I/O（SIO）设备 236 可以耦合于南桥和 I/O 控制器集线器 204。

操作系统在处理器 206 上运行，并且协调和提供图 2 中数据处理系统 200 内的各种组件的控制。操作系统可以是市场上可获得的操作系统，例如 Microsoft® Windows® XP（Microsoft 和 Windows 是微软公司在美国、

其它国家或二者的商标)。面向对象的编程系统，例如 JavaTM 编程系统，可以结合操作系统运行，并且从在数据处理系统 200 上执行的 Java 程序或应用向操作系统提供调用 (Java 和所有基于 Java 的商标是 Sun Microsystems 公司在美国、其它国家或二者的商标)。

用于操作系统的指令、面向对象的编程系统以及应用或程序位于诸如硬磁盘驱动器 226 的存储设备，并且可以被加载到主存储器 208 给处理器 206 执行。说明性实施例的过程可以由处理器 206 使用计算机实现的指令来实现，举例来说，其可以位于诸如主存储器 208、只读存储器 224 的存储器中，或者位于一个或多个外围设备中。

图 1-2 中的硬件可以取决于实现而变化。除了图 1-2 中所描述的硬件之外，或者代替图 1-2 中所描述的硬件，可以使用其它的内部硬件或外围设备，例如闪速存储器、等效非易失性存储器或光盘驱动器等。此外，说明性实施例的过程可以应用于多处理器数据处理系统。

在一些说明性的例子中，数据处理系统 200 可以是个人数字助理 (PDA)，其通常被配置具有闪速存储器，以提供用于存储操作系统文件和/或用户产生的数据的非易失性存储器。总线系统可以包括一个或多个总线，例如系统总线、I/O 总线和 PCI 总线。当然，可以使用在依附于构造或体系结构的不同组件或设备之间提供数据传输的任何类型的通信构造或体系结构来实现总线系统。通信单元可以包括用于发送和接收数据的一个或多个设备，例如调制解调器或网络适配器。举例来说，存储器可以是主存储器 208 或者诸如可以在北桥和存储控制器集线器 202 中找到的高速缓冲存储器。处理单元可以包括一个或多个处理器或 CPU。图 1-2 中所描述的例子以及上述例子并不意味着暗示结构限制。例如，数据处理系统 200 除了采取 PDA 的形式之外，还可以是平板计算机 (tablet computer)、膝上型计算机或电话设备。

说明性实施例提供了基于对处理单元、原始流 (primal stream) 以及用户对输出数据的要求的形式描述而自动创建工作流的过程。该过程能够快速适应新近可用的原始流、处理单元以及其它变化的参数、环境或条件，

而不过度使系统资源有所负担，并且没有人工交互。此外，可以将工作流转换成可以在 Web 服务执行环境中执行的格式。

图 3 描述了依照说明性实施例的流处理的例子。在该例中，当特定股票很可能要超过预定值的时候，将相关结果 350 传递给已经请求被通知的用户。在一个例子中，相关结果是具有相关的股票价格变化的一组股票。在这些说明性例子中，原始流或广播流包括贸易 310、电视新闻 320 以及无线电 330。在所描述的例子中，应用组件包括股票分析 312、运动图像专家组 4 (MPEG-4) 多路分用器 322、图像分析 324、语音到文本 (speech-to-text) 326、文本分析 328、语音到文本 332、文本分析 334，以及多路流连接 340。

流处理应用可以由使用可用的原始流的现有应用组件组成，以便应用组件产生满足用户需求的结果。因而，股票分析 312 接收信息流(贸易 310)并且向多路流连接 340 输出结果。

在该例中，MPEG-4 多路分用器 322 接收广播流(电视新闻 320)，并且向图像分析 324、文本分析 328 和语音到文本 326 输出。而语音到文本 326 向文本分析 328 输出。图像分析 324 和文本分析 328 向多路流连接 340 输出。

语音到文本 332 接收原始流(无线电 330)，并且向文本分析 334 输出。而文本分析 334 向多路流连接 340 输出。多路流连接 340 以相关结果 350 的形式提供输出。

在一个实施例中，可以将流特性编码为以流对象参数化的变数(fluent)和判定(predicate)。在编程中，判定是评估表达并且基于数据的条件提供真或假的回答的陈述。这些条件被表示为关于流特性的逻辑表达式。变数是比判定更一般的函数。变数可以从不同于判定的布尔域的域取值。在文献中变数也被称为函数。将组件描述编码为以输入和输出流对象参数化的动作。动作的前提由输入流上转换的输入端口要求组成，并且动作效应利用与输出端口关联的转换公式计算输出流对象的特性。通过基于共享对应于端口的示例动作参数之间的流对象来标识输入 - 输出端口连接，然后将

由规划系统产生的作为动作序列的计划转换成工作流。

说明性实施例提供了一种用于处理多路流连接的可扩缩的分布式解决方案的计算机实现的方法、装置和计算机可用程序代码。由于很多流相关应用比关键字比较要求更复杂的连接判定，因此考虑诸如同等连接和非同等连接的一般流连接。

说明性实施例针对能够分布式执行多路流连接的相关感知元组路由框架。该分布方案可以观察滑动窗连接语义。滑动窗连接的输出含有满足预先定义的连接判定的、并且在其相应的窗口中被同时显现的相关元组的所有集合。由于相关约束，对于维持滑动窗连接语义来说，分布开销是不可避免的。因此，说明性实施例针对在具有最小分布开销的一组分布式主机中多路流连接算子的工作负荷的最优分布。分布开销指的是使用或过度使用系统资源。例如，分布开销可以包括所消耗的处理器时间、存储空间以及处理输入数据流所需要的网络带宽。

在一个说明性实施例中，对准元组路由（ATR）方法或方案使用流划分实现多路流连接的分布式执行。对准元组路由动态地选择最快的输入流作为主流，并且将其它流的元组路由与主流对准，以符合相关约束。对准元组路由将输入流分成被路由至不同主机用于连接处理的分段。分段是流的一部分或片段。连接处理也被称为相关处理。对准元组路由可以用于保持连接语义。此外，对准元组路由的开销独立于主机的数目。相反，对准元组路由开销仅与滑动窗大小以及从流的速率有关。因此，对准元组路由适用于从流具有低速率并且滑动窗大小不是很大的情况。

在另一说明性实施例中，约束元组路由（CTR）方案在相关约束下分别路由不同流的元组。约束元组路由使用流划分和算子划分二者来分布多路流连接算子。与对准元组路由不同，约束元组路由允许将很大的多路连接划分成由不同主机执行的一组较小的多路连接。约束元组路由方法、过程和算法针对的是利用轻负荷主机的最小集合覆盖相关元组的问题。约束元组路由用于保持连接语义，并且具有独立于主机数目的开销。不同于对准元组路由，约束元组路由的开销独立于滑动窗大小，这使得约束元组路

由更适于具有大滑动窗规格的连接算子。

为支持连续流，流处理系统将滑动窗与每个流相关联。窗口含有称为元组的流上最近到达的数据项。元组是一组值或流数据。流中的数据也称为流数据，并且其是以比特、字、数字或形成于一个或多个数据流的其它流式数据的形式所接收的信息。窗口可以是基于时间的或基于元组的。基于时间的窗口可以是，例如，在最后 10 分钟内到达的元组，而基于元组的窗口可以是，例如，最后 1000 个元组。重要的连续查询算子之一是两个流（流 S_1 和流 S_2 ）之间的滑动窗连接。该窗口连接的输出含有来自于流 S_1 和流 S_2 的每对这样的元组，即其满足连接判定并且在其相应的窗口中被同时显现。

连接判定是关于两个元组之间的一个或多个共同属性的比较函数。基本连接判定是两个元组 s_1 和 s_2 之间关于共同属性 A 的相等比较，表示为 $s_1 \cdot A = s_2 \cdot A$ 。然而，可以将说明性方案应用于任何一般的连接判定。滑动窗连接具有很多应用。例如，考虑两个流，其中一个流含有电话呼叫记录，而另一个流含有股票交易记录。可以使用这样的滑动窗连接来产生交易欺诈警报，即该滑动窗连接运作以便在可疑电话呼叫与异常交易记录之间关于共同属性“贸易标识符”进行相关或连接。

图 4 是依照说明性实施例的滑动窗流连接算子模型的示图。图 4 用于描述滑动窗连接的语义，并且给出集中式加窗连接处理过程。可以在诸如图 1 的服务器 104 和 106 或客户机 110、112 和 114 的服务器或客户机中实现多路流连接算子 400。

多路流连接算子 400 包括表示为 S_i 的各种数据流，包括流 1 402、流 2 404、流 3 406 和流 4 408。数据流由表示为 $s_i \in S_i$ 的一连串元组或数据项组成。每个流可以具有可变的数据到达速率。例如， r_i 表示流 S_i 在当前时间周期的平均到达速率。在动态流环境中，平均流速率 r_i 可以随时间变化。在该例中，每个元组 $s_i \in S_i$ 携带时间戳 $s_i \cdot t$ 来表示元组 s_i 到达流 S_i 的时间。诸如 $S_i[t_1, t_2]$ 的语言表示在时间 $[t_1, t_2]$ 期间到达流 S_i 的所有元组。

为了处理无限数据流，将滑动窗与每个流相关联，用于将连接处理的

范围限制在新近到达的元组。例如， $S_i[W_i]$ 表示流 S_i 上的滑动窗，其中 W_i 表示单位时间的窗口长度。在时间 t ，如果 s_i 在时间间隔 $[t-W_i, t]$ 内到达 S_i ，则 s_i 属于 $S_i[W_i]$ 。因此，可以将 $S_i[W_i]$ 认为是 $S_i[t-W_i, t]$ 的缩写。

将 $n, n \geq 2$ 个输入流中的多路流连接算子 400 表示为 $J_i = S_1[W_1] \bowtie S_2[W_2] \dots \bowtie S_n[W_n]$ 。多路流连接算子 400 的输出包括所有的元组群 (s_1, s_2, \dots, s_n) ，以便在时间 $s_i \cdot t, \forall s_i \in S_i, \forall s_k \in S_k[W_k], 1 \leq k \leq n, k \neq i, s_1, \dots, s_n$ 满足预先定义的连接判定 $\theta(s_1, \dots, s_n)$ 。连接判定可以是诸如多维空间中的距离函数的任何一般的函数。例如，在图 4 中，考虑连接结果，包括在时间 8 到达的流 3 406 的元组 $s_3 410$ ，表示为 $s_3 < 8 > 410$ 。 $s_i < t >$ 表示在时间 t 到达 S_i 的元组。多路流连接算子 400 将流 3 406 的 $s_3 < 8 > 410$ 与在时间 8 的包括在滑动窗 $S_1[W_1], S_2[W_2]$ 和 $S_4[W_4]$ 中的所有元组进行比较。例如，在多路流连接算子 400 中，元组 $s_3 < 8 > 410$ 需要首先与 $S_4[4, 7] 412$ 中的元组连接。

图 5 是依照说明性实施例的分布式多路流连接执行模型的示图。多路流连接经常是资源密集的。例如，多路流连接可以具有对缓冲所有滑动窗中的元组的大存储需求，以及对多个输入流中的大量连接探测的快速处理的需求。单个主机可能由于多路流连接查询处理而易于过载。用于处理多路流连接的可扩缩的分布式流处理系统 500 针对这些问题。可扩缩的分布式流处理系统 500 包括扩散算子 502 OP_d 、融合算子 504 OP_f ，以及通过高速网络 508 而连接的一群服务器主机 506 V_1 。

该服务器主机群集 506 可以包括服务器，例如通过诸如图 1 的网络 102 的网络而互联的服务器 104 和 106。

融合算子 504 和扩散算子 502 是可以在诸如图 2 的数据处理系统 200 的通用计算机上实现的软件模块。扩散算子 502 可以将多路流连接的工作负载加速到多个分布式主机上。融合算子 504 将分散的连接结果融为完整的结果，例如图 3 的相关结果 350。

扩散算子 502 将来自于输入流 512 的输入元组 510 动态路由至不同的服务器主机 506 用于连接处理，而融合算子 504 可以将分散的连接结果 508

融为完整的查询应答 514。与连接算子不同，扩散算子 502 进行简单的元组路由计算，并且要求对输入流 512 的小缓冲。扩散算子 502 的处理时间常常比连接计算的处理时间少几个以上的数量级。因而，扩散算子 502 不是可扩缩的分布式流处理系统 500 的瓶颈。

对分布式连接执行的一个基本要求是保持连接结果的正确性。元组路由方案不应当遗漏任何的连接结果或产生重复的连接结果。然而，强制 (brute-force) 元组路由方案可能违犯多流相关约束或相关约束。在先前的例子中，在图 4 的多路流连接算子 400 中，元组 $s_3<8>$ 410 需要首先与 $S_4[4, 7]$ 412 中的元组连接。通过分布式执行方案可以将图 4 的 $S_4[4, 7]$ 412 中的元组分散到诸如服务器主机 506 的不同主机上。假设元组 $s_4<4>; s_4<5>; s_4<6>$ 位于 v_1 506 上，并且元组 $s_4<5>; s_4<6>; s_4<7>$ 位于 v_2 506 上。如果将 $s_3<8>$ 发送至 v_1 506 或 v_2 506，则遗漏了一些连接结果。如果将 $s_3<8>$ 发送至 v_1 506 和 v_2 506 二者，则可能产生重复的连接结果。为了保持滑动窗连接语义，必须仔细设计元组路由方案以满足相关约束：

定义 1：给定 n 路连接算子 $J_i=S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ ，任何 (s_1, s_2, \dots, s_n) ， $s_i \in S_i$ ， $1 \leq i \leq n$ ，其必须一次且仅一次由出现于相同主机上的 J_i 相关。

在示出分布式多路连接执行需要复制元组或路由中间连接结果以符合相关约束的证明中，令来自于 n 个输入流的元组表示需要被相关的输入流。给定相反的假设，即既未将元组复制于多个主机上，也未将中间结果路由经过不同的主机。考虑每两个连续的元组 $s_i<t_i>$ 和 $s_j<t_j>$ ， $0 \leq t_j - t_i \leq W_i$ ，如果 $i \neq j$ ，那么由于 $s_j<t_j>$ 需要相关 $s_i<t_i>$ ，因此需要将 $s_j<t_j>$ 路由至与 $s_i<t_i>$ 的相同的主机上。如果 $i=j$ ，则由于 $s_j<t_j>$ 需要和与 $s_i<t_i>$ 连接的那些元组连接，因此也需要将 $s_j<t_j>$ 路由至与 $s_i<t_i>$ 的相同的主机上。然后，将 n 个输入流的所有相关的元组都路由至相同的主机，这就成为集中式连接执行。

因而，分布式连接执行需要在不同的主机之间复制元组或路由中间结果，这被称为扩散开销。这些开销元组和路由操作可能消耗处理器时间、存储空间以及系统中的网络带宽。因此，扩散算子的目标是在相关约束下实现最优的分布式连接执行。相关约束在形式上定义如下：

定义 2：给定连接算子 $J_i = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ 和 m 个主机 $\{v_1, \dots, v_m\}$ ，将每个元组最优地路由至一个或多个主机，以便（1）满足相关约束，（2）最优地平衡不同主机的工作负荷，以及（3）最小化扩散开销。

图 6A-6B 是依照说明性实施例的相关感知元组路由方案的示图。图 6A-6B 表示用于多路流连接算子的可扩缩处理的一组相关感知元组路由方案。图 6A-6B 的方案允许单个连接算子利用多个主机 602 的资源，或者允许多个连接算子以细粒度共享多个主机 602 的资源。粒度指示工作负荷划分的大小。较细粒度意味着可以将相关工作负荷分成可以分布于不同主机上的较小的片。主机可以是多个互联的服务器，例如图 1 的服务器 104 和 106。

任务是计算机系统承担以实现预定目标的过程。多路流相关工作负荷是计算机资源的总量，例如处理需求、存储器，以及在多个输入流上进行相关处理所需要的带宽。连续优化过程是连续地调整计算机系统的操作，以在动态计算环境中实现最优性能，从而解决连续优化问题。诸如相关处理的连接处理包括计算机系统需要承担以便将一个流的元组与所有其它流的元组相关的一组操作。

在高级别，该组相关感知元组路由方案以二维完成分布式多路流连接执行：（1）流划分：将输入流分成被路由至不同主机 602 的分段；以及（2）算子划分：将多路连接算子分成利用不同主机 602 上处理的相关来计算的分算子。

图 6A 是单独进行流划分的称为对准元组路由（ATR）的简单相关感知元组路由方案。图 6A 描述了使用四个主机 602 执行三路连接算子的对准元组路由方案 600 的分布快照（distribution snapshot）。为了符合相关约束，对准元组路由方案 600 调整所有输入流的元组路由，并且复制流划分的边界周围的元组子集。对准元组路由方案 600 是不经过不同主机 602 路由中间结果的单跳（one-hop）路由过程。每个主机 602 仅仅只是在所有输入元组的子集上进行整个连接计算。

例如，在图 6A 中，主机 602 v_1, v_2, v_3 和 v_4 分别在于时间 $[t, t+T], [t+T,$

$[t+2T]$ 、 $[t+2T, t+3T]$ 和 $[t+3T, t+4T]$ 内到达的元组上执行连接计算。然而，如果多路连接算子将很多输入流与大的滑动窗相关，则对准元组路由的分布粒度可能会太粗糙，导致低效率的资源利用和大的复制开销。

称为约束元组路由（CTR）方案 604 的图 6B 的第二元组路由方案探究流划分和算子划分二者。不同于对准元组路由方案 600，约束元组路由方案 604 通过允许经过不同的主机 602 路由中间连接结果 606，可以分离多个主机 602 中的多路连接计算的执行。因此，约束元组路由方案 604 可以减少复制开销，因为对准元组路由系统不必保证在第一路由跳转中所有相关的元组都位于相同的主机上。

例如，在图 6B 中，约束元组路由方案 604 首先连接主机 v_1 上的元组 S_1 和 S_2 ，并且然后将中间结果路由至主机 v_2 以连接第三个流 S_3 。相比于对准元组路由方案 600，约束元组路由方案 604 具有路由中间结果 606 的额外开销，但却可以以较细粒度实现较好的负荷分布。在图 6B 中，约束元组路由方案 604 通过允许将一个多路连接计算分成可以在不同主机上执行的多个两路连接来进行算子分割。

下面在图 11 和图 14 中进一步解释了对准元组路由方案、方法和算法的细节。下面在图 12 和图 15 中进一步解释了约束元组路由方案、方法和算法的细节。

图 7 是依照说明性实施例的对准元组路由模型的示图。图 7 给出了诸如图 6A 的对准元组路由方案 600 的对准元组路由方案的设计细节。对准元组路由模型 700 同时相互配合地路由连接算子的输入流的元组。对准元组路由模型 700 使用流划分实现分布式流连接执行。对准元组路由模型 700 包括主流 702、从流 704 和从流 706。 V_1 指示主机， W_1 指示滑动窗大小，并且 T 指示分段长度。主流 702 是具有最高数据流速率的流。当主流 702 的速率变的慢于从流之一的时候，对准元组路由采用转换阶段（transition phase）来改变主流 702。

对准元组路由是一种用于解决连续优化问题的方案。对准元组路由动态地选择一个输入流作为主流，并且根据时间戳将所有其它输入流的元组

与主流对准。相比之下，为了符合相关约束，基于对主流的划分将其它的流划分成重叠的分段，其被称为从流 704 和 706。

通常，对准元组路由连续地将所有输入流分成不同的分段，每个分段含有在特定时间周期内到达的元组。基于相关约束将从流 704 和 706 的分段与主流 702 的分段对准。将属于已对准的分段的元组路由至相同的主机用于产生连接结果。

图 8 是依照说明性实施例的约束元组路由模型的示图。图 8 给出了包括诸如图 6B 的约束元组路由方案 604 的约束元组路由方案的设计细节的约束元组路由（CTR）模型 800。约束元组路由模型 800 是一种独立地路由不同流的元组的方案。对于流 1 802、流 2 804 和流 3 806 的每个输入流来说，约束元组路由模型 800 基于所有先前的相关元组的布局进行路由判定。

约束元组路由是另一种用于解决连续优化问题的方案。约束元组路由模型 800 分别路由不同输入流的元组，而不是像对准元组路由方法那样同时路由来自不同输入的元组。图 8 示出了用于三路流连接算子的约束元组路由方案。对于具有探测序列 $s_i \bowtie S_{i1} \bowtie \dots \bowtie S_{in-1}$ 的任何元组 $s_i \in S_i$, $1 \leq i \leq n$ ，约束元组路由基于先前的相关元组的布局为元组 s_i 以及所有的中间连接结果 $x_i = s_i \bowtie S_{i1} \dots \bowtie S_{ik}$, $1 \leq k \leq n-1$ 进行路由判定。

为了避免要求所有的连接算子进行路由计算，将约束元组路由实现为计算 s_i 的整个路由路径从而与其它 $n-1$ 个流连接的源路由过程。每个元组携带其路由路径以表示其需要访问来产生连接结果的主机的集合。为了减少路由计算开销，约束元组路由将每个输入流上的元组分组成段，并且将每个分段作为整体路由至不同的主机。因而，约束元组路由仅需要计算每个分段的路由而不是每个元组的路由。分段长度表示负荷平衡粒度与路由开销之间的折衷。

约束元组路由还维护记录先前所路由的分段的布局的路由表。如果分段不需要基于多路流连接语义而与任何未来的分段相关，则从路由表删除分段信息。

图 9 是依照说明性实施例的集中式连接方法的流程图。图 9 的方法可以在诸如图 4 的多路流连接算子 400 的多路流连接算子中实现。

通过接收用于所连接的流的元组开始过程(步骤 902)。例如，步骤 902 的原始数据流可以由输入流缓冲器接收。步骤 902 的原始流可以是诸如图 4 的流 1 402、流 2 404、流 3 406 和流 4 408 的流。

接下来，该过程选择一个流以基于时间戳处理(步骤 904)。例如，根据当前所缓冲的元组的时间戳，该流是下一元组 s_i 。接下来，该过程根据所选择的流的时间戳标记所有其它流中的过期元组(步骤 906)。该过程接下来从所有的流移除已被所有其它的流处理并且被标记为过期的过期元组(步骤 908)。步骤 906 和 908 用于查找步骤 904 中所选择的流的连接顺序。

接下来，该过程基于连接顺序产生连接结果(步骤 910)。该过程然后确定流连接是否是完整的(步骤 912)。如果流连接不是完整的，则该过程接收用于所连接的流的元组(步骤 902)。在步骤 910 中，该过程还可以更新指针 p_i 以指出步骤 904 中将要选择的、在输入流缓冲器 Q_i 中的下一个流或元组。如所示，重复该过程以继续处理所有的流。如果在步骤 912 的确定中流连接是完整的，则该过程结束。

图 10 是依照说明性实施例的多路流连接方法的流程图。图 10 的方法可以在诸如图 5 的可扩缩的分布式流处理系统 500 的多路流连接系统中实现。

通过接收用于所连接的流的元组开始过程(步骤 1002)。该过程然后在扩散算子处为每个元组计算路由路径(步骤 1004)。接下来，该过程基于由扩散算子计算的路由路径将元组路由至一个或多个主机(步骤 1006)。接下来，该过程在不同的主机并行地进行集中式连接(步骤 1008)。主机可以是诸如图 5 的服务器主机 506 的服务器主机。集中式连接可以包括图 9 的方法和步骤。

接下来，该过程在融合算子处基于连接标识聚集连接结果(步骤 1010)。融合算子可以是诸如图 5 的融合算子 504 的算子。该过程然后确定流连接是否是完整的(步骤 1012)。如果流连接是完整的，则该过程终止。

如果流连接在步骤 1012 不是完整的，则该过程接收用于所连接的流的元组（步骤 1002）。当已经处理了所有输入元组的时候，就已完全产生了完整的连接结果。

图 11 是依照说明性实施例的对准元组路由方法的流程图。可以使用诸如图 7 的对准元组路由模型 700 的对准元组路由模型实现图 11 中所描述的步骤。连续地重复图 11 的过程来处理输入元组。

通过接收元组 S_i 开始过程（步骤 1102）。接下来，该过程确定元组是否属于主流 S_A （步骤 1104）。步骤 1104 的确定基于元组的流标识以及主流的标识。如果元组属于主流，则该过程确定是否开始新的分段（步骤 1106）。步骤 1106 的确定基于元组的时间戳以及当前分段的开始/结束时间。如果确定开始新的分段，则该过程存储最后选择的主机 V_b^{last} （步骤 1110）。接下来，该过程选择新的主机 V_b （步骤 1112）。该过程然后将 S_i 发送至新选择的主机 V_b （步骤 1114）。此后，该过程将更新的分段开始时间更新为 $t=t+T$ （步骤 1116）。接下来，该过程确定流连接是否是完整的（步骤 1109）。

返回步骤 1106，如果该过程确定不开始新的分段，则该过程将 S_i 发送至为当前分段选择的主机（步骤 1108）。接下来，该过程确定流连接是否是完整的（步骤 1109）。如果该过程确定流连接是完整的，则该过程终止。如果在步骤 1109 中该过程确定流连接不是完整的，则该过程接收元组 S_i （步骤 1102）。

返回步骤 1104，如果在步骤 1104 中该过程确定元组并不属于主流 S_A ，则该过程确定是否开始新的分段（步骤 1118）。步骤 1118 的确定基于元组的时间戳以及当前分段的开始/结束时间。如果该过程确定不开始新的分段，则该过程首先将 S_i 发送至为当前分段选择的主机 V_b （步骤 1120）。接下来，该过程确定 S_i 是否在 $t+W_A$ 之前到达（步骤 1122）。如果 S_i 确实是在 $t+W_A$ 之前到达的，则该过程将 S_i 发送至为最后的分段选择的主机 V_b^{last} （步骤 1124）。接下来，该过程确定流连接是否是完整的（步骤 1109）。

如果在步骤 1122 中 S_i 不是在 $t+W_A$ 之前到达的，则该过程确定流连接是否是完整的（步骤 1109）。

如果在步骤 1118 中该过程确定开始新的分段，则该过程将 $S_i[t+T-W_i, t+T]$ 冲刷 (flush) 至 V_b (步骤 1126)。接下来，该过程将 S_i 发送至 V_b 和 V_b^{last} (步骤 1128)。接下来，该过程将分段开始时间更新为 $t=t+T$ (步骤 1130)。接下来，该过程确定流连接是否是完整的 (步骤 1109)。图 14 的伪代码中进一步解释了图 11 的过程。

图 12A-12B 是依照说明性实施例的约束元组路由方法的流程图。可以使用诸如图 8 的约束元组路由模型 800 的约束元组路由模型实现图 12A-12B。

通过接收元组 S_i 开始过程 (步骤 1202)。接下来，该过程确定是否开始新的分段 (步骤 1204)。如果该过程确定开始新的分段，则该过程检索探测序列 (步骤 1206)。接下来，该过程以 $v_0=\emptyset$ 初始化第一路由跳转 (步骤 1208)。接下来，该过程设置 $k=1$ (步骤 1210)。

接下来，该过程确定是否 $k < n$ (步骤 1214)。如果 $k > n$ ，则该过程更新路由表路径 (步骤 1216)。接下来，该过程将分段开始时间更新为 $t=t+T$ (步骤 1218)。接下来，该过程将新的分段的位置信息添加到路由表中 (步骤 1220)。接下来，该过程确定流连接是否是完整的 (步骤 1221)。如果流连接是完整的，则该过程终止。如果在步骤 1221 中流连接不是完整的，则该过程接收元组 S_i (步骤 1202)。

如果在步骤 1214 中 $k < n$ ，则该过程检索 $S_{ik}[W_{ik}]$ 中分段的位置 (步骤 1222)。接下来，该过程移除由前一跳 V_{k-1} 所覆盖的那些分段 (步骤 1224)。接下来，该过程为 $S_{ik}[W_{ik}]$ 计算最小的集合覆盖 (步骤 1226)。接下来，该过程为避免重复而注释路由路径 (步骤 1228)。接下来，该过程将 V_k 附加到路由路径 P (步骤 1230)。接下来，该过程设置 $k=k+1$ (步骤 1232)。该过程然后返回确定是否 $k < n$ (步骤 1214)。

返回步骤 1204，如果在步骤 1204 中该过程确定不开始新的分段，则该过程检查路由表以获得 S_i 的当前分段的路由路径 (步骤 1234)。接下来，该过程以路由路径注释 S_i (步骤 1236)。接下来，该过程将 S_i 的副本发送至第一路由跳转中的每个主机 (步骤 1238)。接下来，该过程确定流连接

是否是完整的（步骤 1239）。如果流连接是完整的，则该过程终止。如果在步骤 1239 中流连接不是完整的，则该过程接收元组 S_i （步骤 1202）。图 15 的伪代码中进一步解释了图 12A-12B 的过程。

图 13 是依照说明性实施例的多路流连接算法。多路流连接算法 1300 是用于在单个主机上处理多路流查询 $S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ 的集中式算法。多路流连接算法 1300 可以通过诸如图 4 的多路流连接算子 400 的算子实现。在诸如图 9 的步骤的过程中可以实现多路流连接算子的基本步骤。系统对每一个输入流 S_i 维持一个队列 Q_i , $1 \leq i \leq n$, 用于缓冲输入元组。 Q_i 可以是输入缓冲器。当新的元组 $s_i \in S_i$ 到达的时候，如果本地主机上的存储空间没有满，则将新的元组插入相应的队列 Q_i 。

否则，系统要么舍弃新近到达的元组，要么用新近到达的元组替换缓冲器中旧的元组。以时间顺序处理所有队列中的元组。例如，如果 $s_i \cdot t < s_j \cdot t$, 那么首先处理 s_i 。每个队列 Q_i 都保持指针指向由连接算子当前处理的元组的缓冲器中的元组。如果当前正在处理的元组是 s_i ，则连接算子将 s_i 与所有其它的流 $S_j[W_j]$, $1 \leq j \leq n$, $j \neq i$ 进行比较，以产生包括 s_i 的所有的连接结果。

第 j 个流的滑动窗 $S_j[W_j]$ 包括在时间 $s_i \cdot t - W_k$ 与 $s_i \cdot t$ 之间到达 S_j 的所有元组 s_j 。两个元组之间的每个连接判定评估称为一个连接探测。

基于不同的流之间的连接选择性动态地判定 $s_i \in S_i$ 的连接顺序[11, 1, 10]。对 s_i 的连接处理从其自身开始，并且选择与 S_i 具有最小选择性的流 S_i 作为下一跳。然后，利用与 S_i 具有最少连接选择性的下一个所选择的流 S_k 连接所有的中间结果 $s_i \bowtie S_j[W_j]$ 。例如，在图 4 中，当前正在处理的元组是在时间 8 到达流 S_3 的 $s_3 < 8 > 410$ 。 S_3 的探测序列是 $S_3 \rightarrow S_4[W_4] \rightarrow S_1[W_1] \rightarrow S_2[W_2]$ 。因而，基于 S_3 与 S_4 之间的连接判定 $\theta_{3,4}$ 首先将 s_2 与 $S_4[W_4]$ 进行比较。将中间结果 $s_3 \bowtie S_4[W_4]$ 与 $S_1[W_1]$ 进行比较。最后，将 $s_3 \bowtie S_4[W_4] \bowtie S_1[W_1]$ 与 $S_2[W_2]$ 进行比较。

图 14 是依照说明性实施例的对准元组路由算法。对准元组路由算法 1400 有差别地处理连接算子 $J = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ 的输入流，并且根据一

个选择的主流来对准所有流的元组路由操作。对准元组路由算法 1400 可以在诸如图 7 的其中 S_1 是主流的、用于三路流连接算子的对准元组路由模型 700 的路由模型中实现。对准元组路由算法的基本步骤可以在诸如图 11 的步骤的过程中实现。

对准元组路由算法 1400 动态地选择一个输入流作为主流，表示为 S_A ，并且根据时间戳将所有其它输入流的元组与主流对准。扩散算子将主流 S_A 切成或划分成不相交的分段。不相交的分段是彼此不具有共同元组的分段。将一个分段中所有的元组路由至相同的主机，而基于诸如最小负荷最先 (LLF) 的特定的调度策略将不同的分段路由至不同的主机。相比之下，为了符合相关约束，将其它 $n-1$ 个流基于对主流的划分而划分成重叠的分段，其被称为从流。重叠的分段是彼此具有至少一个共同元组的分段。

通常，对准元组路由连续地将所有输入流分成不同的分段，每个分段含有在特定的时间周期内到达的元组。 $S_i[t, t+T]$ 表示包括在时间 $[t, t+T]$ 内到达 S_i 的所有元组的 S_i 的分段，其中 t 称为分段的开始时间，并且 T 称为分段长度。基于相关约束将从流的分段与主流的分段对准。将属于已对准的分段的元组路由至相同的主机用于产生连接结果。例如，图 7 示出了用于主流 S_1 和两个从流 S_2 、 S_3 的对准元组路由算法的路由结果。为了便于说明，假定 $r_i=1$ 元组/秒， $i=1, 2, 3$ ，并且分段长度 $T=5$ ，并且三个滑动窗的大小 $W_1=2$ ， $W_2=2$ 和 $W_3=3$ 。扩散算子将主流 S_1 分成被分别路由至主机 v_1 、 v_2 和 v_3 的不相交的分段。将从流 S_2 划分成重叠的分段： $S_2[1, 7]$ 到 v_1 、 $S_2[4, 12]$ 到 v_2 ，以及 $S_2[9, 17]$ 到 v_3 。将从流 S_3 也划分成重叠的分段： $S_3[1, 7]$ 到 v_1 、 $S_3[3, 14]$ 到 v_2 ，以及 $S_3[8, 17]$ 到 v_3 。

图 14 描述了用于扩散连接算子 $S_1[W_1] \bowtie S_2[W_2] \dots \bowtie S_n[W_n]$ 的负荷的具体的对准元组路由算法 1400 步骤。图 14 示出了用于使用 m 个主机 $\{v_1, \dots, v_m\}$ 处理 J 的对准元组路由算法 1400 的伪代码。区段 1402 描述了用于主流 S_A 的路由步骤。

当扩散算子接收到来自于 S_A 的元组 s_A 的时候，其首先根据 s_A 的时间戳 $s_A \cdot t$ 检查 s_A 是否属于当前分段 $S_A[t, t+T]$ 。如果 $t \leq s_A \cdot t < t+T$ ，则 s_A 属

于当前分段，并且将其路由至在分段的开始时间 t 所选择的主机 v_i 。如果 $s_A \cdot t \geq t+T$ ，则对准元组路由开始新的分段，并且选择新的主机作为该新的分段的路由目的地。对准元组路由遵循最小负荷最先（LLF）策略来为每个分段选择主机。

因为在分布式连接处理系统中考虑了诸如处理器、存储器和网络带宽的不同资源，所以组合度量 w_i 表示主机 v_i 的负荷条件。对于每种资源类型 R_i ，对准元组路由算法 1400 定义负荷指示器 $\Phi_{Ri} = \frac{U_{Ri}}{C_{Ri}}$ ，其中 U_{Ri} 和 C_{Ri} 分别表示主机 v_i 上的资源 R_i 的使用和容量。将负荷值 w_i 定义如下：

$$w_i = \omega_1 \Phi_{cpu} + \omega_2 \cdot \Phi_{memory} + \omega_3 \cdot \Phi_{bandwidth}$$

其中 $\sum_{i=1}^3 \omega_i = 1, 0 \leq \omega_i \leq 1$ 表示可以由系统动态配置的不同资源类型的重要性。

基于对主流的划分，对准元组路由将所有的从流分成重叠的分段用于保持相关约束。对于从流 S_i , $i \neq A$, 如果对准元组路由将分段 $S_A[t, t+T]$ 路由至主机 v_k ，则对准元组路由将分段 $S_i[t-W_i, t+T+W_A]$ 路由至相同的主机 v_k ，以便符合相关约束。类似地，如果对准元组路由将主流的下一分段 $S_A[t+T, t+2T]$ 发送至主机 v_j ，则对准元组路由需要将从流的分段 $S_i[t+T-W_i, t+2T+W_A]$, $1 \leq i \leq n, i \neq A$ 发送至相同的主机 v_j 。因而，将在时间周期 $[t+T-W_i, t+T+W_A]$ 之间到达 S_i 的元组发送至 v_i 和 v_j 二者。重复元组的数目是 $r_i \cdot (W_A + W_i)$ 。

对准元组路由算法 1400 假设扩散算子具有缓冲容量以高速缓冲 $S_i[t-W_i, t], 1 \leq i \leq n, i \neq A$ 中的元组。如果该假设不成立，则将对每个分段 $S_A[t, t+T]$ 的主机选择移位至较早的时间 $t-W_j$ ，其中 W_j 表示所有从流中最大的滑动窗。例如，在图 7 中，在时间 $t=3$ 选择对第二分段的布局。然后，对准元组路由将 $S_2[4, 7]$ 和 $S_3[3, 7]$ 中的元组路由至主机 v_1 和 v_2 二者。

通过示出对准元组路由与原始连接算子产生相同的连接结果集合证明了对准元组路由算法 1400 的正确性。 $C(J)$ 和 $C'(J)$ 分别表示由原始连接算子 $J = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ 以及使用对准元组路由算法的分布式处理方案所产生的连接结果的集合。通过示出 $C(J) = C'(J)$ 证明了对准元组路由算法的正确性。

定理 A: 给定多路流连接算子 $J=S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$, 令 $C(J)$ 和 $C'(J)$ 分别表示由原始连接算子以及由使用对准元组路由算法的分布式处理方案所产生的连接结果的集合。由此, $C(J) = C'(J)$ 。

证明概略: 通过示出 $\forall s_i, 1 \leq i \leq n$ 证明 $C(J) \subseteq C'(J)$, 如果 $s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{in-1}[W_{in-1}] \in C(J)$, 那么 $s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{in-1}[W_{in-1}] \in C'(J)$ 。这是通过示出如果对准元组路由将 s_i 发送至服务器 v_i , 则对准元组路由也将 $\forall s_{ik} \in S_{ik}[W_{ik}], 1 \leq k \leq n-1$ 发送至 v_i 来证明的。

首先考虑 s_i 属于主流的情况。假设将 $s_i \in S_i[t, t+T]$ 发送至主机 v_i 。对准元组路由也将 $S_{ik}[t-W_{ik}, t+T+W_i]$ 发送至 v_i 。另一方面, 滑动窗 $S_{ik}[W_{ik}]$ 包括 $S_{ik}[s_i \cdot t - W_{ik}, s_i \cdot t]$ 中所有的元组。因为 $s_i \cdot t \in [t, t+T]$, 所以 $S_{ik}[s_i \cdot t - W_{ik}, s_i \cdot t] \subseteq S_{ik}[t-W_{ik}, t+T+W_i]$ 也是正确的。因而, 对准元组路由也将 $S_{ik}[W_{ik}], 1 \leq k \leq n-1$ 中所有的元组发送至主机 v_i 。

当考虑 s_i 属于从流的时候, 证明 $\forall s_A \in S_A[W_A]$, 其中 S_A 表示主流, 对准元组路由在 s_A 被发送的机器上发送 s_i 的副本。假设 s_A 属于分段 $S_A[t, t+T]$, 并且被发送至 v_i 。对准元组路由也将分段 $S_i[t-W_i, t+T+W_A]$ 发送至 v_i 。通过证明 $s_i \in S_i[t-W_i, t+T+W_A]$, 则因为 s_A 属于分段 $S_A[t, t+T]$, 所以得到 $t \leq s_A \cdot t < t+T$ 。因而, 结果是 $s_i \cdot t \geq s_A \cdot t \geq t$ 以及 $s_i \cdot t < s_A \cdot t + W_A < t+T+W_A$ 。因而, 结果是 $t \leq s_i \cdot t < t+T+W_A$ 。从而 s_i 属于分段 $S_i[t-W_i, t+T+W_A]$, 其也被发送至 v_i 。

通过证明 $\forall s_j \in S_j[W_j]$, 其中 S_j 表示从流, 对准元组路由将 s_i 的副本和 s_j 发送至相同的主机。假设 s_i 属于与被路由至 v_i 的主流分段 $S_A[t, t+T]$ 对准的分段 $S_i[t, t+T]$ 。因而, 对准元组路由也将 $S_j[t-W_j, t+T+W_A]$ 发送至 v_i 。接下来, 通过证明 $S_j[W_j] = S_j[s_i \cdot t - W_j, s_i \cdot t] \subseteq S_j[t-W_j, t+T+W_A]$, 则因为 $t \leq s_i \cdot t < t+T$, 得出 $t-W_j \leq s_i \cdot t - W_j$ 以及 $s_i \cdot t < t+T+W_A$ 。

因而, $S_j[W_j] \subseteq S_j[t-W_j, t+T+W_A]$ 也被发送至 v_i 。得出结论, 即 $\forall s_i, s_i$ 和 $S_{ik}[W_{ik}], 1 \leq k \leq n-1$ 出现在相同的主机上。因而, $C(J) \subseteq C'(J)$ 。

接下来证明 $C'(J) \subseteq C(J)$ 。首先, 由扩散连接算子所产生的 $C'(J)$ 中的任何连接结果都遵循多路流连接语义, 这也应当出现在 $C(J)$ 中。其

次，由于主流上的任何元组 $\forall s_A \in S_A$ 不会出现在两个不同的主机上，所以对准元组路由并不产生重复的连接结果。因而， $C'(J) \subseteq C(J)$ 。结合 $C(J) \subseteq C'(J)$ 和 $C'(J) \subseteq C(J)$ ，得到结果 $C(J) = C'(J)$ 。□

还可以分析对准元组路由算法的开销。相比于原始输入流，由于从流的部分重复，对准元组路由将更多的元组推入了系统中。将对准元组路由算法的开销定义为每单位时间由对准元组路由产生的额外元组的数目。分布式流处理系统需要消耗一部分网络带宽、CPU 时间以及存储空间，用于传输、处理以及缓冲那些开销数据。

定理 B: 给定多路流连接算子 $J = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ ，令 S_A 表示当前的主流。令 T 表示分段长度。令 r_i , $1 \leq i \leq n$ 表示流 S_i 的平均速率。令 O_{ATR} 表示对准元组路由算法的平均开销。由此， $O_{ATR} = \sum_{i=1, i \neq A}^n \frac{W_i + W_A}{T}$ 。

证明概略：对于在时间周期 T 上的每个分段 $S_A[t, t+T]$ ，以及每个从流 S_i , $1 \leq i \leq n$, $i \neq A$ ，对准元组路由引入比原始从流 S_i 更多的元组 $r_i \cdot (W_i + W_A)$ 。因而，对于每个分段长度 T ，由对准元组路由算法产生的额外元组的总数是 $\sum_{i=1, i \neq A}^n r_i \cdot (W_i + W_A)$ 。因而，每单位时间由对准元组路由算法产生的额外元组的平均数目是 $\sum_{i=1, i \neq A}^n \frac{r_i \cdot (W_i + W_A)}{T}$ 。□

以上分析揭示了对准元组路由算法的有趣的特性。对准元组路由算法的开销独立于用于处理连接算子的主机数。该特性使得对准元组路由特别适合于大规模流处理群集，其中可以从大量主机搜集可用资源而没有过多的负荷扩散开销。

各种自适应方案可以用于优化动态流环境中对准元组路由算法的性能。根据定理 B，观察到对准元组路由算法的开销与分段长度成反比。由于在较长的时间周期上分摊了重叠分段的成本，因此较大的分段长度引入较少的负荷扩散开销。然而，由于大分段中的大量元组被强制到相同的主机，因此大的分段长度限制了负荷平衡粒度。

因此，对准元组路由自适应地调整或触发分段长度 T 的分段自适应，以便在动态流环境中维持最优的性能。对准元组路由采用基于采样的仿形 (profiling) 过程来找到当系统条件改变时的最优分段长度。令 T 表示当

前的分段长度，并且 ΔT 表示自适应步长值。自适应过程测试 $T+\Delta T$ 和 $T-\Delta T$ 二者。如果 $T+\Delta T$ 的性能较好，则最优分段长度应当大于当前的分段长度。系统逐渐增加分段长度，直到所测量的系统性能达到其峰值。否则，如果 $T-\Delta T$ 产生较好的性能，则系统逐渐降低分段长度以搜索最优值。对准元组路由总是在一个分段的末尾改变分段长度，以保证自适应不违反相关约束。

对准元组路由的开销仅与从流的速率有关，而独立于主流的速率。在动态流环境中，每个输入流的速率可以随时间动态地改变。因此，对准元组路由根据定理 B 动态地选择具有最小负荷扩散开销的主流。主流应当总是具有最高速率的流。当主流的速率变得慢于从流之一的时候，对准元组路由采用转换阶段来改变主流。类似于分段自适应，总是在一个分段的末尾触发流角色交换以符合相关约束。

图 15 是依照说明性实施例的约束元组路由算法。约束元组路由算法 1500 可以在诸如图 8 的约束元组路由模型 800 的路由模型中实现。约束元组路由算法 1500 的基本步骤可以在诸如图 12A-B 的步骤的过程中实现。

约束元组路由 1500 在区段 1502 中为具有探测序列 $s_i \bowtie S_{i1} \bowtie \dots \bowtie S_{in-1}$ 的任何元组 $s_i \in S_i$, $1 \leq i \leq n$ 进行路由判定，基于先前的相关元组的布局，约束元组路由为元组 s_i 以及所有的中间连接结果 $x_i = s_i \bowtie S_{i1} \dots \bowtie S_{ik}$, $1 \leq k \leq n-1$ 进行路由判定。为了避免要求所有的连接算子进行路由计算，将约束元组路由作为计算 s_i 的整个路由路径以便与其它 $n-1$ 个流连接的源路由算法来实现。每个元组携带其路由路径以表示其需要访问来产生连接结果的主机的集合。

为了减少路由计算开销，约束元组路由将每个输入流上的元组分组成段，并且将每个分段作为整体路由至不同的主机。因而，约束元组路由仅需要计算每个分段的路由而不是每个元组的路由。分段长度表示负荷平衡粒度与路由开销之间的折衷。约束元组路由还维护记录先前所路由的分段的布局的路由表。如果分段不需要基于多路流连接语义与任何未来的分段相关，则从路由表删除该分段的信息。

在区段 1504 中，约束元组路由为需要与滑动窗 $S_{il}[W_{il}]$ 中的元组连接

的分段 $\eta_i = S_i[t, t+T]$ 进行路由判定。约束元组路由首先得到 $S_{i1}[W_{i1}] = S_{i1}[t-W_{i1}, t+T]$ 中所有元组的位置。为最小化开销，约束元组路由选择可以覆盖所有的相关元组的最小主机集合 $V_1 \subseteq \{v_1, \dots, v_m\}$ 。将以上问题用公式表达为将详细描述的加权最小集合覆盖问题。在约束元组路由期间，连接顺序用于将多路相关处理分成多个较小的算子。具体地，将 n 路连接算子 $s_i \bowtie S_{i1} \bowtie \dots \bowtie S_{in-1}$ 划分成 $(n-1)$ 个 2 路连接算子 $x_{i1} = s_i \bowtie S_{i1}$ ， $x_{i2} = x_{i1} \bowtie S_{i2}$ ， \dots ， $x_{in-1} = x_{in-2} \bowtie S_{in-1}$ ，每个 2 路连接算子都可以在不同的主机上执行。在每跳查找相关分段的位置。在每跳计算覆盖所有相关分段的最小主机集合。约束元组路由然后将 η_i 的第一路由跳转设置为 V_1 中的所有主机。可以将分段 $\eta_i = S_i[t, t+T]$ 保存在 V_1 中每个主机上的存储缓冲器 Q_i 中，直到根据连接语义不再需要其元组。

约束元组路由还更新路由表以记录分段 $\eta_i = S_i[t, t+T]$ 位于 V_1 中的一组主机上。例如，在图 8 中，约束元组路由计算 $S_1[9, 10]$ 的路由，其探测序列是 $s_1 \rightarrow S_2[W_2] \rightarrow S_3[W_3]$ 。约束元组路由得到 $S_2[W_2]$ 中所有相关元组的布局： $S_2[1, 2]$ 在 v_1, v_2 上； $S_2[3, 4]$ 在 v_2, v_3 上； $S_3[5, 6]$ 在 v_1, v_4 上，等等。约束元组路由然后选择可以覆盖 $S_2[W_2]$ 中所有元组的最小主机集合 $V_1 = \{v_2, v_4\}$ 。

因此，约束元组路由将 $S_1[9, 10]$ 的路由路径上的第一跳设置为 $V_1 = \{v_2, v_4\}$ 。约束元组路由还在路由表中添加条目，指定将分段 $S_1[9, 10]$ 置于主机 $V_1 = \{v_2, v_4\}$ 。

接下来，约束元组路由需要将中间结果 $s_i \bowtie S_{i1}[W_{i1}]$ 路由至覆盖 $S_{i2}[W_{i2}]$ 中所有元组的主机。类似于第一步，约束元组路由首先获取 $S_{i2}[W_{i2}]$ 中所有相关元组的位置。然而，为了最小化经过不同主机传输中间结果的开销，对 $s_i \bowtie S_{i1}[W_{i1}]$ 的路由判定应当考虑 $s_i \bowtie S_{i1}[W_{i1}]$ 的当前位置。给定第一路由跳转 $V_1 = \{v_1, \dots, v_k\}$ ，约束元组路由首先消除 $S_{i2}[W_{i2}]$ 中已经被 V_1 中的主机覆盖的那些元组。合理性在于当前位于 V_1 中的主机上的任何中间结果 $s_i \bowtie S_{i1}[W_{i1}]$ 应当与 $S_{i2}[W_{i2}]$ 中局部可用的元组连接。然后，约束元组路由计算最小主机集合 V_2 以覆盖 $S_{i2}[W_{i2}]$ 中那些剩余的元组。然而，与原始元组

不同，并不将中间结果缓冲于存储队列用于与其它元组连接。因而，约束元组路由并不需要将中间结果的布局记录在路由表中。

例如，在图 8 中，第二路由跳转是为当前位于主机 v_2, v_4 的中间结果 $S_2[9, 10] \bowtie S_2[W_2]$ 选择一组主机。然后，由于 $S_3[3, 4]$ 和 $S_3[7, 8]$ 已经被主机 v_2 覆盖，因此约束元组路由将其移除。接下来，基于剩余元组的位置，例如在 $\{v_3, v_5\}$ 上的 $S_3[1, 2]$ 、在 $\{v_5, v_6\}$ 上的 $S_3[5, 6]$ 、在 $\{v_6, v_7\}$ 上的 $S_3[9, 10]$ ，约束元组路由计算最小主机集合 $V_2 = \{v_5, v_6\}$ 作为分段 $S_1[9, 10]$ 的第二路由跳转。

重复以上计算直到约束元组路由计算出 $s_i \bowtie S_{i1} \bowtie \dots \bowtie S_{in-1}$ 中所有 $n-1$ 个探测步骤的主机集合 V_1, \dots, V_{n-1} 。然后，约束元组路由通过为分段 η_i 插入条目来更新路由表，其中分段 η_i 的位置是由 V_1 所指定的主机集合。在区段 1504 中，约束元组路由算法 1500 以路由路径 $V_1 \rightarrow V_2 \dots \rightarrow V_{n-1}$ 注释每个元组 $s_i \in S_i[t, t+T]$ 。图 14 示出了用于使用 m 个主机 $\{v_1, \dots, v_m\}$ 处理连接算子 $J_i = S_1[W_1] \bowtie S_2[W_2] \dots \bowtie S_n[W_n]$ 的约束元组路由算法的伪代码。

另一算法可以用于诸如图 12A-B 的过程的最优主机集合选择。最优主机选择算法的目的是为每个路由跳转 V_k , $1 \leq k \leq n-1$ 选择最佳主机集合。第 k 个路由跳转的目标是在 $X_{k-1} = s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{ik-1}[W_{ik-1}]$, $x_o = s_i$ 与 $S_{ik}[W_{ik}]$ 中所有的元组之间产生所有的连接结果。假设滑动窗 $S_{ik}[W_{ik}]$ 包括表示为 $E = \{\eta_1, \dots, \eta_e\}$ 的一组分段。从路由表检索每个分段的布局信息。

每个分段 η_z , $1 \leq z \leq e$ 分布在主机集合 $U_z \subseteq \{v_1, \dots, v_m\}$ 上。约束元组路由然后将分段布局信息转换成主机覆盖信息。例如，假设 η_z 分布在主机集合 U_z 上， U_z 中的每个主机覆盖分段 η_z 。让我们表示为 $Y = \bigcup_{z=1}^e U_z$ 。对于每个主机 $v_i \in Y$ ，其覆盖形成 E 的子集的一组分段，表示为 $A_i \subseteq E$ 。因为目标是实现平衡的负荷分布，所以将 X_{k-1} 分布于可以覆盖 $S_{ik}[W_{ik}]$ 中所有相关元组的最少数目的最少负荷主机。因而，权值 w_i 与每个子集 A_i 相关联。权值 w_i 是主机 v_i 的负荷值 w_i ，其由负荷值的方程来定义。因此，将最优主机选择问题用公式表达为加权最小集合覆盖问题：

定义：给定基本集合（ground set） $E = \{\eta_1, \dots, \eta_e\}$ ，子集 $A_1, \dots,$

$A_k \subseteq E$, 以及每个子集 A_i 的成本 w_i , 目标是找到最小集合覆盖 $I \subseteq \{1, \dots, K\}$, 以便 $\cup_{j \in I} A_j = E$ 以及 $\sum_{j \in I} w_j$ 最小。

根据 I 导出主机集合 V_k 。例如, 如果集合覆盖 $I = \{1, 2\}$, 那么 $V_k = \{v_1, v_2\}$ 。最小集合覆盖问题是众所周知的 NP 难解问题 (NP-hard problem)。因此, 约束元组路由使用贪婪启发式 (greedy heuristic) 算法来找到最小集合覆盖。基本思想是选择具有 $\operatorname{argmin}_j, A_j \neq 0 \frac{w_j}{|A_j|}$ 的最小值的子集 A_j , 其中 $|A_j|$ 表示集合 A_j 的基数。将 A_j 添加到集合覆盖 I 中, 并且通过移除包括在 A_j 中的那些元素来更新每个剩余的子集。重复将 A_j 添加到集合的过程, 直到所选择的集合覆盖 I 包括 $E = \{\eta_1, \dots, \eta_e\}$ 中所有的分段。

然而, 以上方案可能进行多余的连接计算。假设当前所选择的主机集合是 V_k 。对于任何分段 $\eta_z \in S_{ik}[W_{ik}]$, 其被置于主机集合 $U_z = \{v_{z1}, \dots, v_{z1}\}$ 上。如果集合 V_k 和 U_z 含有多于一个的共同主机 (即, $|V_k \cap U_z| > 1$), 则在含于 $|V_k \cap U_z|$ 中的不同主机处多余地计算了 X_{k-1} 与 η_z 之间的连接探测。这样的多余计算可能会导致多余的连接结果。为了解决该问题, 注释由 X_{k-1} 的不同副本所携带的路由路径, 以保证仅由一个主机执行每个连接探测。为了与 $S_{ik}[W_{ik}]$ 中所有的元组相关, 将 X_{k-1} 的副本发送至 V_k 中的所有主机。对于位于 V_z 中的主机上的 $\forall \eta_z \in \{\eta_1, \dots, \eta_e\}$, 如果 $|V_k \cap U_z| > 1$, 则选择来自于 $V_k \cap U_z$ 的最小负荷主机 V_j 执行 X_i 与 η_z 之间的连接探测。对于任何其它的主机 $v_j \in V_k \cap U_z$, 以标志 (v_j / η_z) 注释路由路径, 其意味着 $s_i \bowtie s_{i1}[W_{i1}] \dots \bowtie s_{ik-1}[W_{ik-1}]$ 的任何中间结果元组并未与主机 v_j 上的 η_z 连接。

通过证明约束元组路由与原始连接算子产生相同的连接结果的集合, 示出了约束元组路由算法的正确性。 $C(J)$ 和 $C'(J)$ 分别表示由原始连接算子 $J = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ 以及由使用约束元组路由算法的分布式处理方案所产生的连接结果的集合。通过示出 $C(J) = C'(J)$, 证明了约束元组路由算法的正确性。

定理 C: 给定多路流连接算子 $J = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$, 令 $C(J)$ 和 $C'(J)$ 分别表示由原始连接算子以及由使用约束元组路由算法的分布式处理方案所产生的连接结果的集合。由此, $C(J) = C'(J)$ 。

证明概略：通过示出如果 $\forall s_i, 1 \leq i \leq n, s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{in-1}[W_{in-1}] \in C(J)$ ，那么 $s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{in-1}[W_{in-1}] \in C'(J)$ ，首先证明了 $C(J) \subseteq C'(J)$ 。这是通过证明更强的命题来证明的： $\forall k, 1 \leq k \leq n-1, X_k = s_i \bowtie \dots \bowtie S_{ik}[W_{ik}]$ 是由约束元组路由产生的。使用数学归纳法：(1) 证明当 $k=1$ 时命题成立。由于约束元组路由将 s_i 发送至覆盖 $S_{i1}[W_{i1}]$ 中所有元组的主机集合 V_1 ，因此约束元组路由产生 $s_i \bowtie S_{i1}[W_{i1}]$ ；(2) 假设该命题对于某个 $k, 1 \leq k \leq n-2$ 是成立的，证明该命题对于 $k+1$ 是成立的。根据该假设，约束元组路由产生 $X_k = s_i \bowtie \dots \bowtie S_{ik}[W_{ik}]$ 。由于 $S_{ik+1}[W_{ik+1}]$ 中所有的元组要么与 X_k 共置 (co-located)，要么由第 $k+1$ 个路由跳转 V_{k+1} 中的主机所覆盖，因此 $X_{k+1} = X_k \bowtie S_{ik+1}[W_{ik+1}]$ 中所有的结果元组都是由约束元组路由产生的。因此， $C(J) \subseteq C'(J)$ 。

接下来，证明 $C'(J) \subseteq C(J)$ 。首先，由扩散连接算子所产生的 $C'(J)$ 中的任何连接结果都遵循多路流连接语义，这也应当出现在 $C(J)$ 中。其次，证明约束元组路由并不产生任何重复的结果。由于约束元组路由实现重复避免，因此任何结果元组 $X_k = s_i \bowtie \dots \bowtie S_{ik}[W_{ik}], 1 \leq k \leq n-1$ 都仅由单个主机产生一次。因而， $C'(J) \subseteq C(J)$ 。结合 $C(J) \subseteq C'(J)$ 和 $C'(J) \subseteq C(J)$ ，得到结果 $C(J) = C'(J)$ 。□

将约束元组路由算法 1500 的开销定义为每单位时间由约束元组路由所产生的额外数据元组的数目。不同于进行单跳路由的对准元组路由，约束元组路由进行多跳路由，其不仅需要在多个主机上复制原始输入流的元组，而且需要经过不同主机传输中间结果。

因而，约束元组路由的开销由两部分组成：(1) 通过将分段 $s_i[t, t+T]$ 发送至由第一路由跳转 V_1 所指定的多个主机来复制原始输入流；(2) 将中间结果 $X_k = s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{ik}[W_{ik}]$ 传输至由 V_k 所指定的多个主机。对于开销的第一部分，系统需要花费额外的处理、存储器以及网络带宽，用于那些开销数据。然而，由于并未将中间结果存储在存储缓冲器中，因此中间结果的开销仅引起 CPU 和带宽花费。

定理 D： 给定多路流连接算子 $J = S_1[W_1] \bowtie \dots \bowtie S_n[W_n]$ ，令 $r_i, 1 \leq i \leq n$

表示流 S_i 的平均速率。令 T 表示分段长度。 S_i 的探测序列表示为 S_{i1}, \dots, S_{in-1} 。令 $\sigma_{i,j}$ 定义 S_i 与 S_j 之间的连接选择性。将 $S_i, 1 \leq i \leq n$ 中元组的复制品的平均数目表示为 M_i 。将中间结果 $s_i \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{ik}[W_{ik}]$ 的复制品的平均数目表示为 M_{ik} 。令 O_{CTR} 表示原始数据流的平均复制开销。令 O^*_{CTR} 表示中间结果的平均开销。由此， $O^*_{CTR} = \sum_{i=1}^n \sum_{k=2}^{n-2} M_{ik} \left(\sigma_{i,i} \cdot \prod_{j=1}^{k-1} \sigma_{i,j, i, j+1} \right) \left(r_i \cdot \prod_{j=1}^{k-1} r_j W_{ij} \right)$ 。

证明概略：对于每个分段 $S_i[t, t+T], 1 \leq i \leq n$ ，在 T 的时间周期内，约束元组路由相比于原始输入流发送 $(M_i - 1) \cdot r_i \cdot T$ 个额外元组。因而，每单位时间由约束元组路由算法所产生的额外元组的平均数目是 $\sum_{i=1}^n (M_i - 1) \cdot r_i$ 。对于每个分段 $S_i[t, t+T], 1 \leq i \leq n$ ，其需要与 $S_{i1}[W_{i1}], \dots, S_{in-1}[W_{in-1}]$ 连接。从 $S_i[t, t+T] \bowtie S_{i1}[W_{i1}]$ 产生的中间结果的数目是 $\sigma_{i,i_1} (r_i \cdot T) \cdot (r_{i_1} \cdot W_{i_1})$ 。将每个中间结果发送至 M_{i2} 以连接 $S_{i2}[W_{i2}]$ 。用于 $S_i[t, t+T] \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{ik}[W_{ik}]$ 的中间结果的开销是 $M_{ik} \cdot \sigma_{i,i_1} (r_i \cdot T) \cdot (r_{i_1} \cdot W_{i_1}) \cdot \sigma_{i_1,i_2} (r_{i_1} \cdot W_{i_1}) \dots \sigma_{i_{k-1},i_k} (r_{i_{k-1}} \cdot W_{i_k})$ 。因而，用于计算 $S_i[t, t+T] \bowtie S_{i1}[W_{i1}] \dots \bowtie S_{in-1}[W_{in-1}]$ 的中间结果的总数是 $\sum_{k=2}^{n-2} M_{ik} \cdot \sigma_{i,i_1} (r_i \cdot T) \cdot (r_{i_1} \cdot W_{i_1}) \cdot \sigma_{i_1,i_2} (r_{i_1} \cdot W_{i_1}) \dots \sigma_{i_{k-1},i_k} (r_{i_{k-1}} \cdot W_{i_k})$ 。对于所有的 n 个输入流，每单位时间由约束元组路由算法产生的中间结果的总数是 $\sum_{i=1}^n \sum_{k=2}^{n-2} M_{ik} \left(\sigma_{i,i_1} \cdot \prod_{j=1}^{k-1} \sigma_{i_j, i_{j+1}} \right) \left(r_i \cdot \prod_{j=1}^k r_{i_j} W_{ij} \right)$ 。□

类似于对准元组路由方案，约束元组路由的开销也独立于用于执行多路流连接算子的主机 $\{v_1, \dots, v_m\}$ 的总数。因而，约束元组路由允许连接算子利用分布式流处理系统中所有可用的主机而没有过多的开销。约束元组路由的开销取决于两个新的参数 M_i 和 M_{ik} ，其定义了用于路由原始元组和中间结果元组的主机集合的平均数目。由于我们的最优主机集合选择算法总是选择最小的主机集合来符合相关约束。 M_i 或 M_{ik} 的值常常远小于总的主机数。不同于对准元组路由的复制开销 O_{CTR} ，原始流的复制开销 O_{CTR} 独立于滑动窗大小。

因此，当连接算子采用大的滑动窗的时候，约束元组路由可以具有比对准元组路由少的开销。尽管相比于对准元组路由，约束元组路由具有额外的中间结果开销，然而由于在实际应用中连接选择性常常很小，因此中间结果开销 O^*_{CTR} 并不显著。对准元组路由与约束元组路由之间的其它的

不同在于对准元组路由有差别地处理 n 个输入流，而约束元组路由平等地处理所有的输入流。因而，对准元组路由更适于在具有小滑动窗的一个快流和一组慢流中连接的情况，而约束元组路由则在所有的输入流都具有类似的速率并且连接算子采用大的滑动窗的时候效果最好。

此外，约束元组路由需要维护路由表，该路由表记录最近所路由的分段的位置。尽管分段长度不影响约束元组路由的复制开销，然而分段长度决定路由表的大小以及路由计算开销。

因而，说明性实施例提供了一种用于在流处理环境中自动规划的方法。当应用于流处理规划问题的时候，所描述的搜索方法实现了相比于其它规划方法显著改善的可扩缩性。通过实现对复杂多路流连接的精确处理，改善了可扩缩性。通过使用并行处理加速了处理。另外，说明性实施例适应数据流波动。

本发明可以采取全硬件实施例、全软件实施例或者既含有硬件元素又含有软件元素的实施例的形式。在优选的实施例中，以软件实现本发明，其包括但不限于固件、常驻软件、微码等。

此外，本发明可以采取可访问于计算机可用或计算机可读介质的计算机程序产品的形式，该计算机可用或计算机可读介质提供由计算机或任何指令执行系统使用的或者与计算机或任何指令执行系统相连的程序代码。对于该描述来说，计算机可用或计算机可读介质可以是可以容纳、存储、通信、传播或传送由指令执行系统、装置或设备使用的或者与指令执行系统、装置或设备相连的程序的任何实体装置。

介质可以是电子、磁性、光学、电磁、红外或半导体系统（或装置或设备）或者传播介质。计算机可读介质的例子包括半导体或固态存储器、磁带、可移动计算机磁盘、随机访问存储器（RAM）、只读存储器（ROM）、硬磁盘和光盘。光盘的当前的例子包括压缩磁盘-只读存储器（CD-ROM）、压缩磁盘-读/写（CD-R/W）和DVD。

适于存储和/或执行程序代码的数据处理系统将包括通过系统总线直接或间接耦合于存储元件的至少一个处理器。存储元件可以包括在程序代

码的实际执行期间所使用的局部存储器、大容量存储器，以及为了减少在执行期间必须从大容量存储器检索代码的次数而提供至少一些程序代码的临时存储的高速缓冲存储器。

输入/输出或 I/O 设备（包括但不限于键盘、显示器、指点设备等）可以直接地或者通过插入 I/O 控制器耦合于系统。

还可以将网络适配器耦合于系统，从而使得数据处理系统能够适于通过介入专用或公用网络耦合于其它的数据处理系统或远程打印机或存储设备。调制解调器、电缆调制解调器和以太网卡仅仅是几种当前可用类型的网络适配器。

已经出于说明和描述的目的给出了对说明性实施例的描述，但并不旨在以所公开的形式穷举或限制本发明。对本领域的普通技术人员来说，很多修改和变形是显而易见的。所选择和描述的实施例是为了最好地解释本发明的原理、实际应用，以及使本领域的普通技术人员能够针对各种实施例以及适于预期的特定用途的各种修改理解本发明。

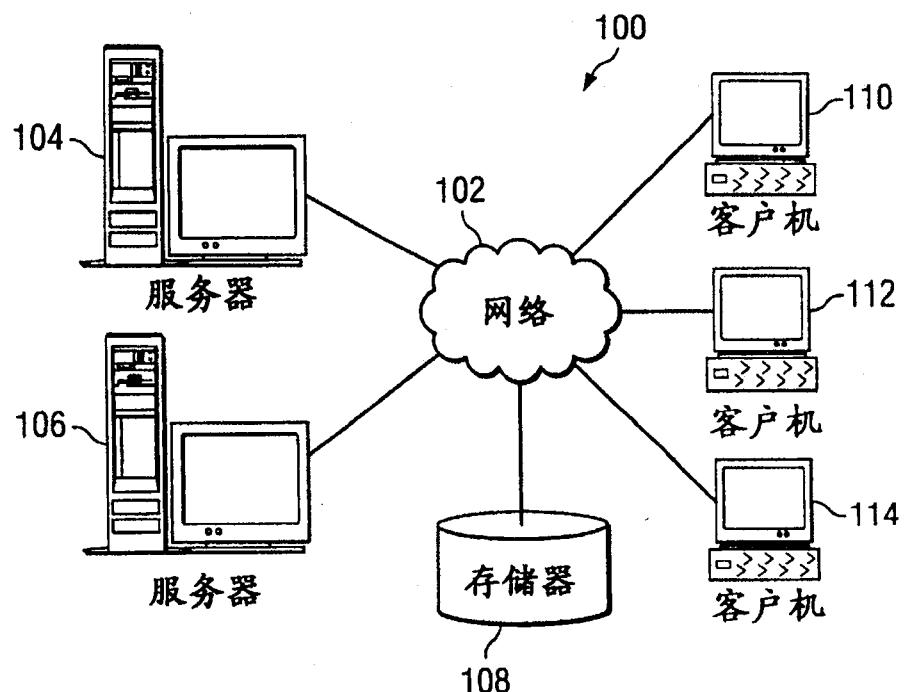


图 1

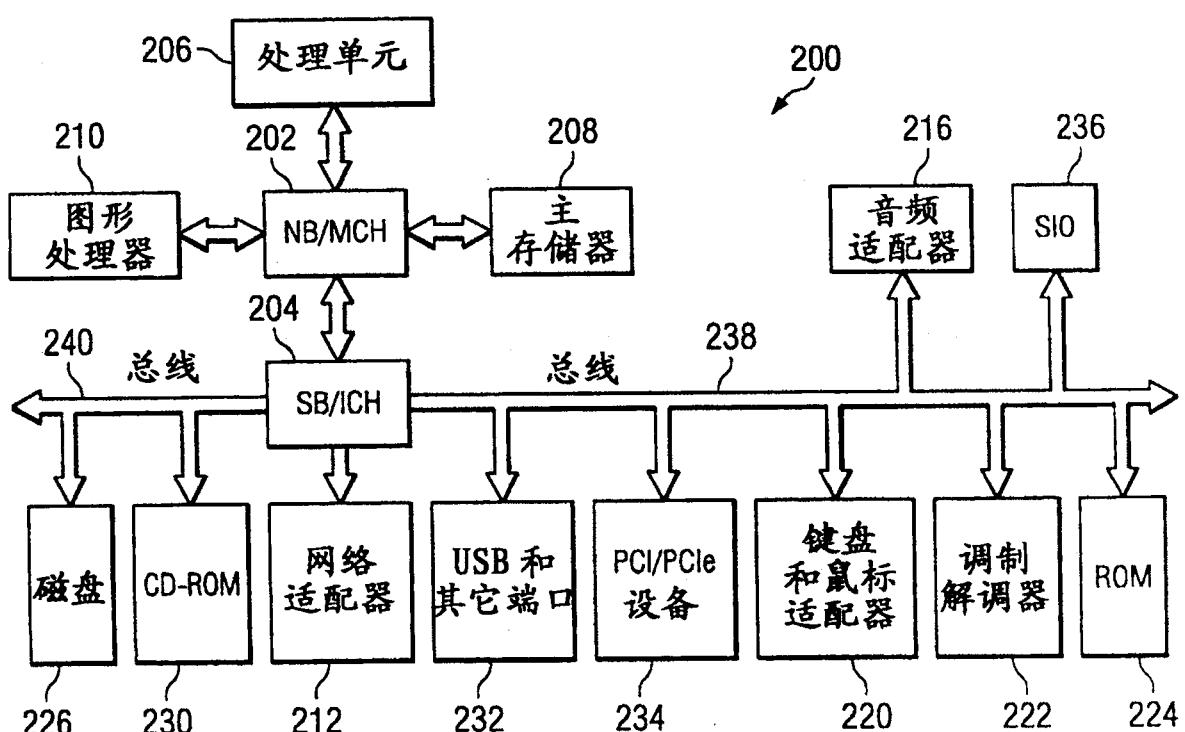


图 2

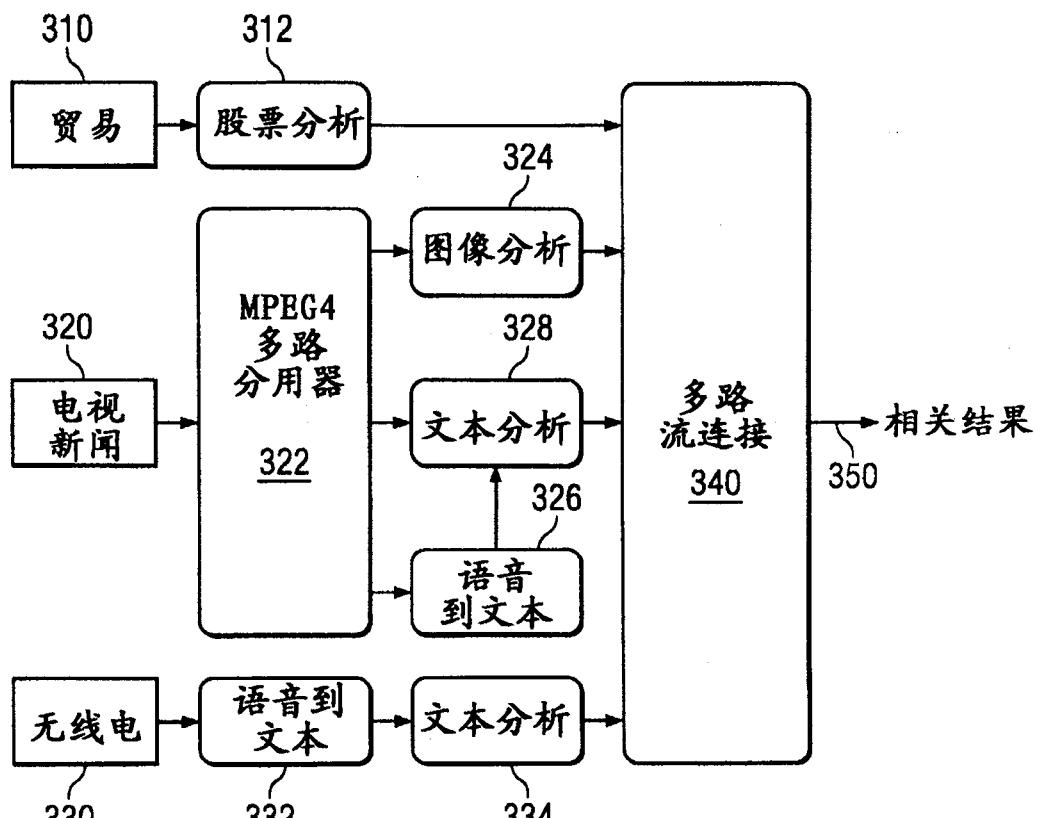
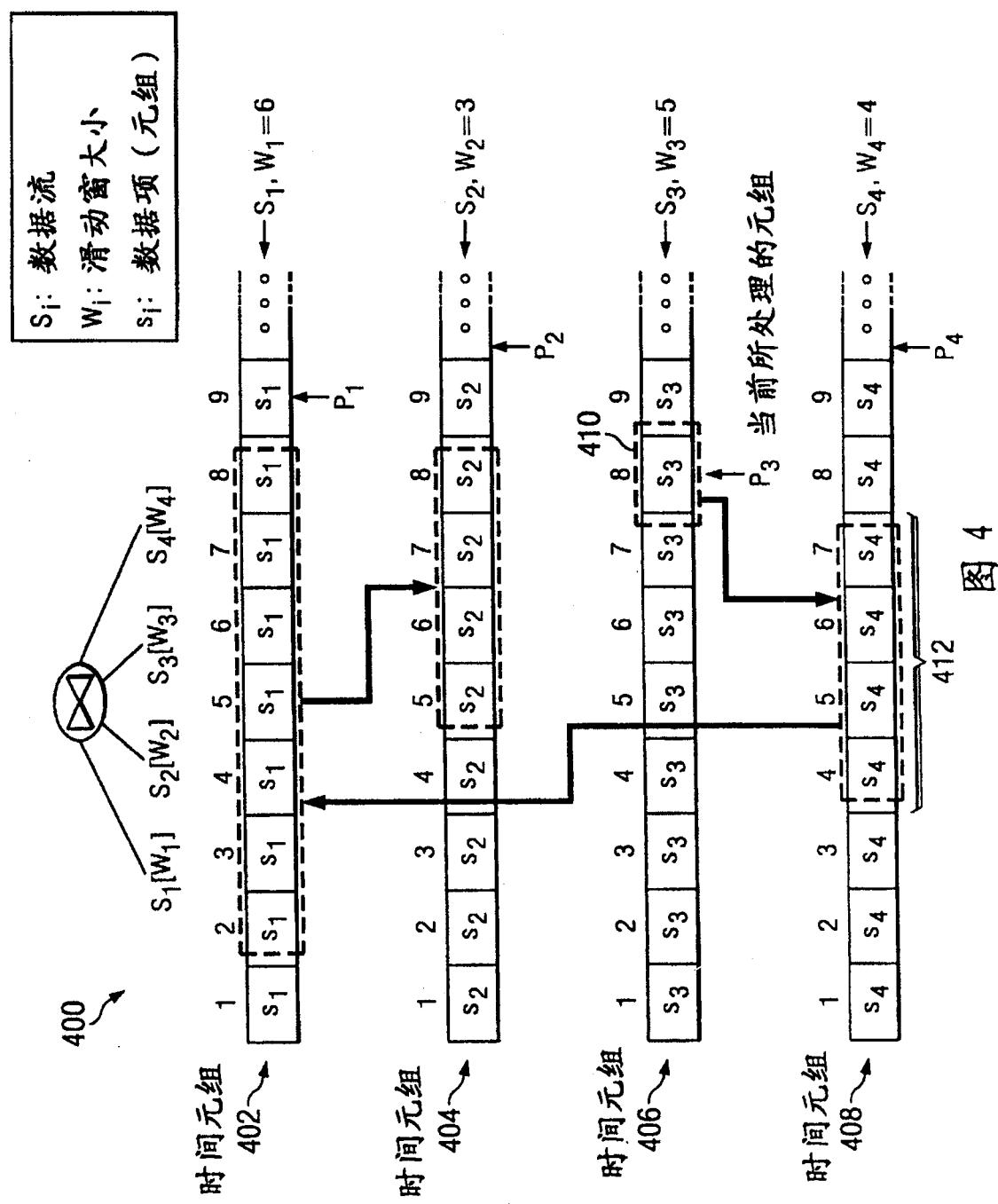


图 3



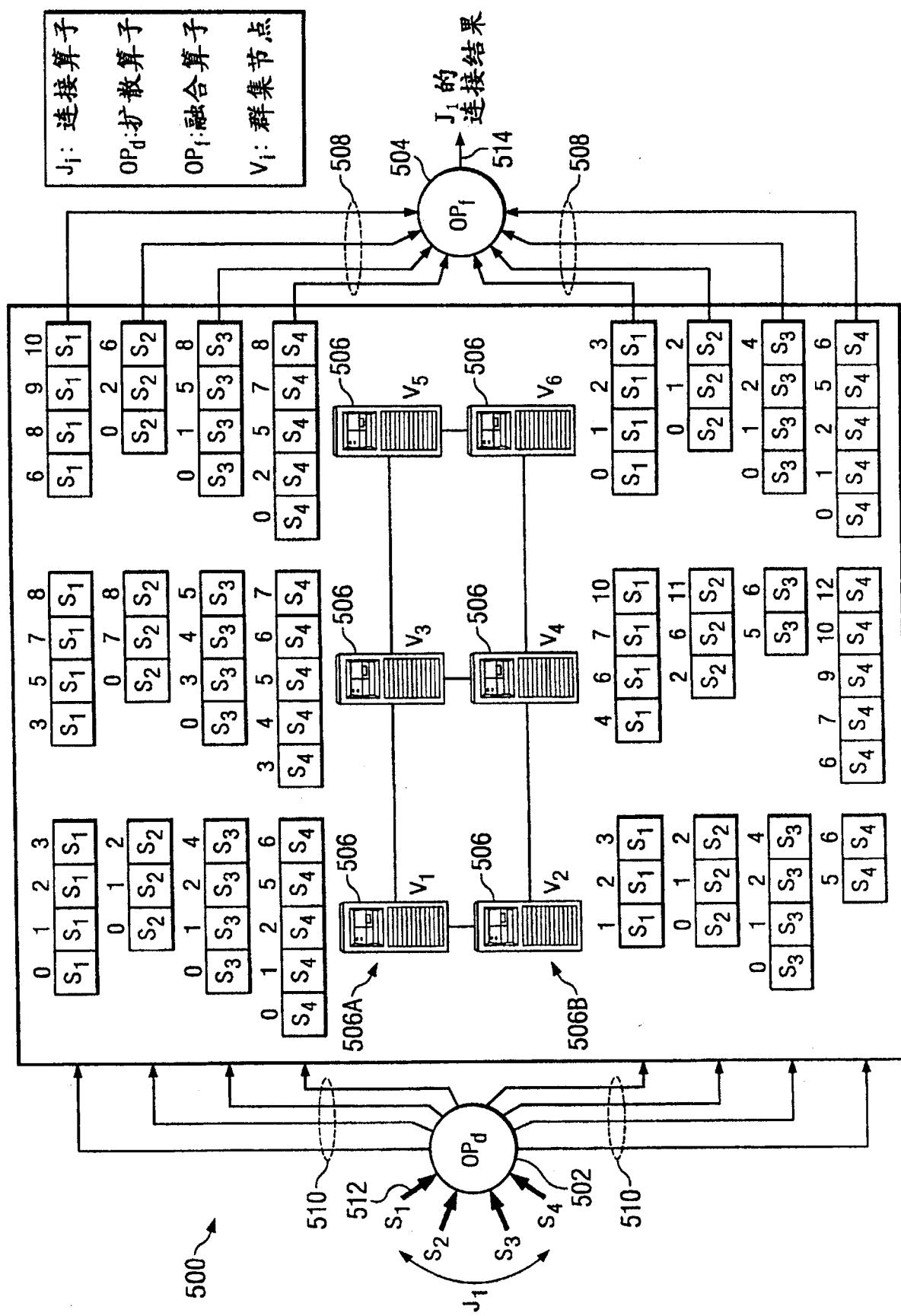


图 5

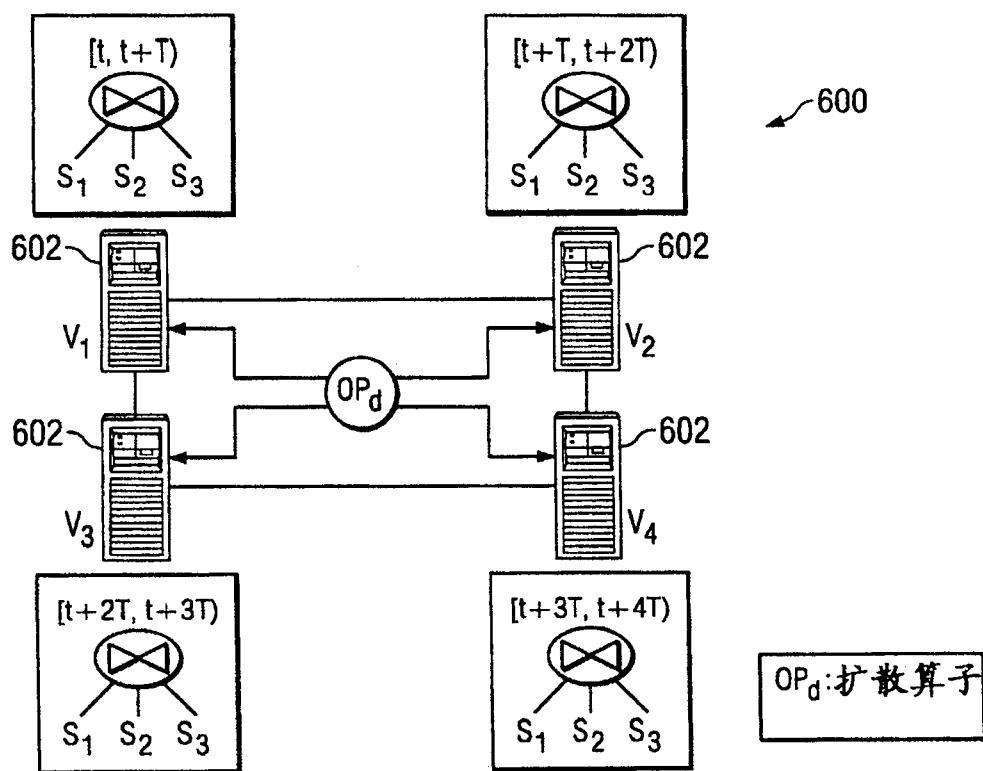


图 6A

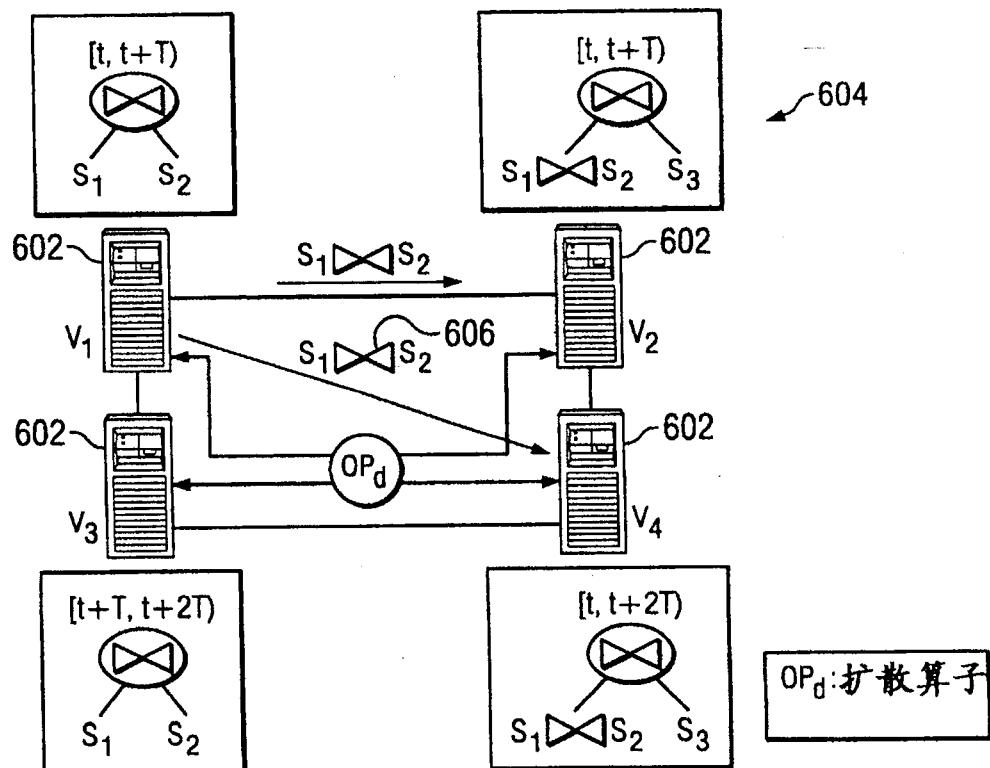
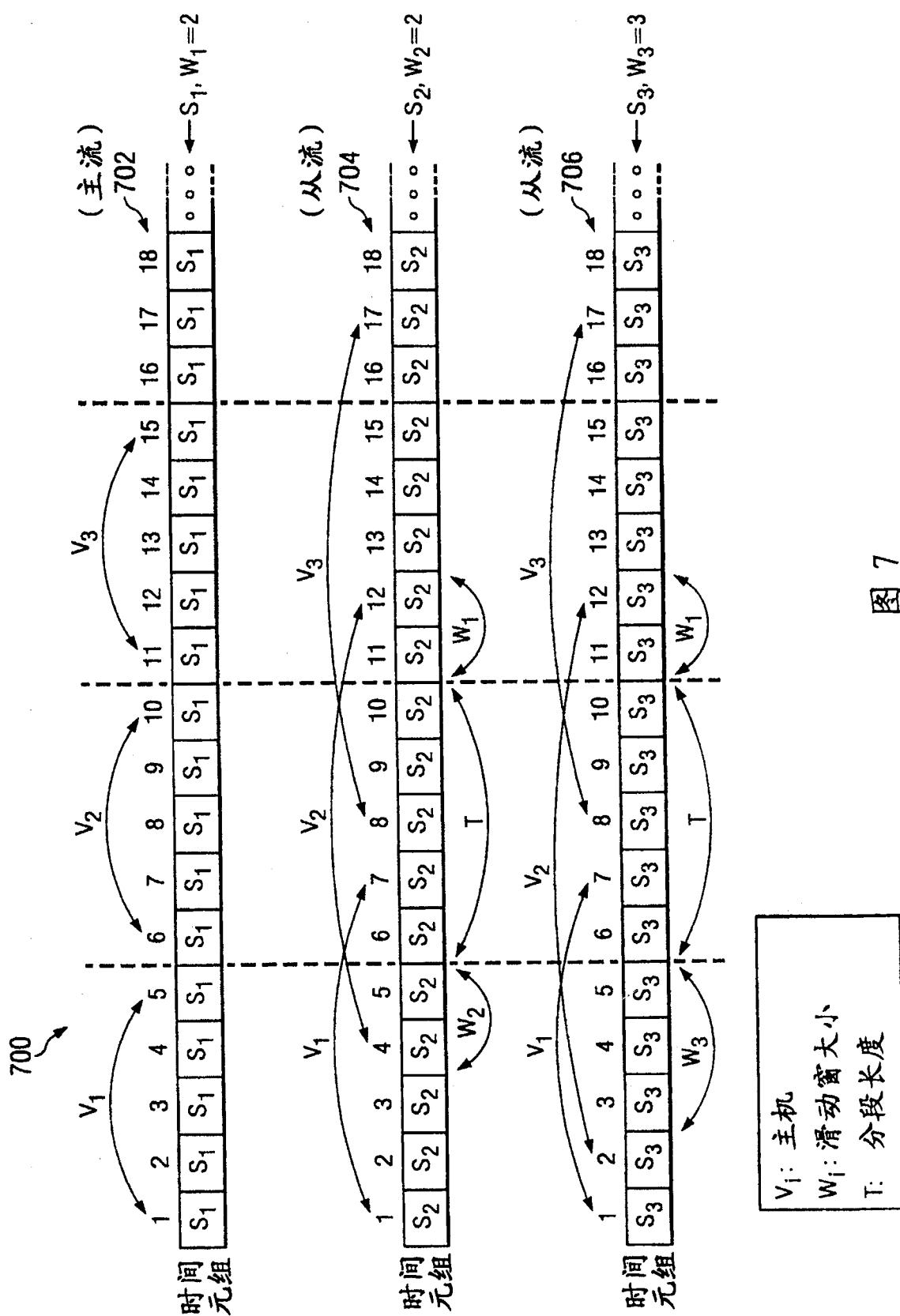


图 6B



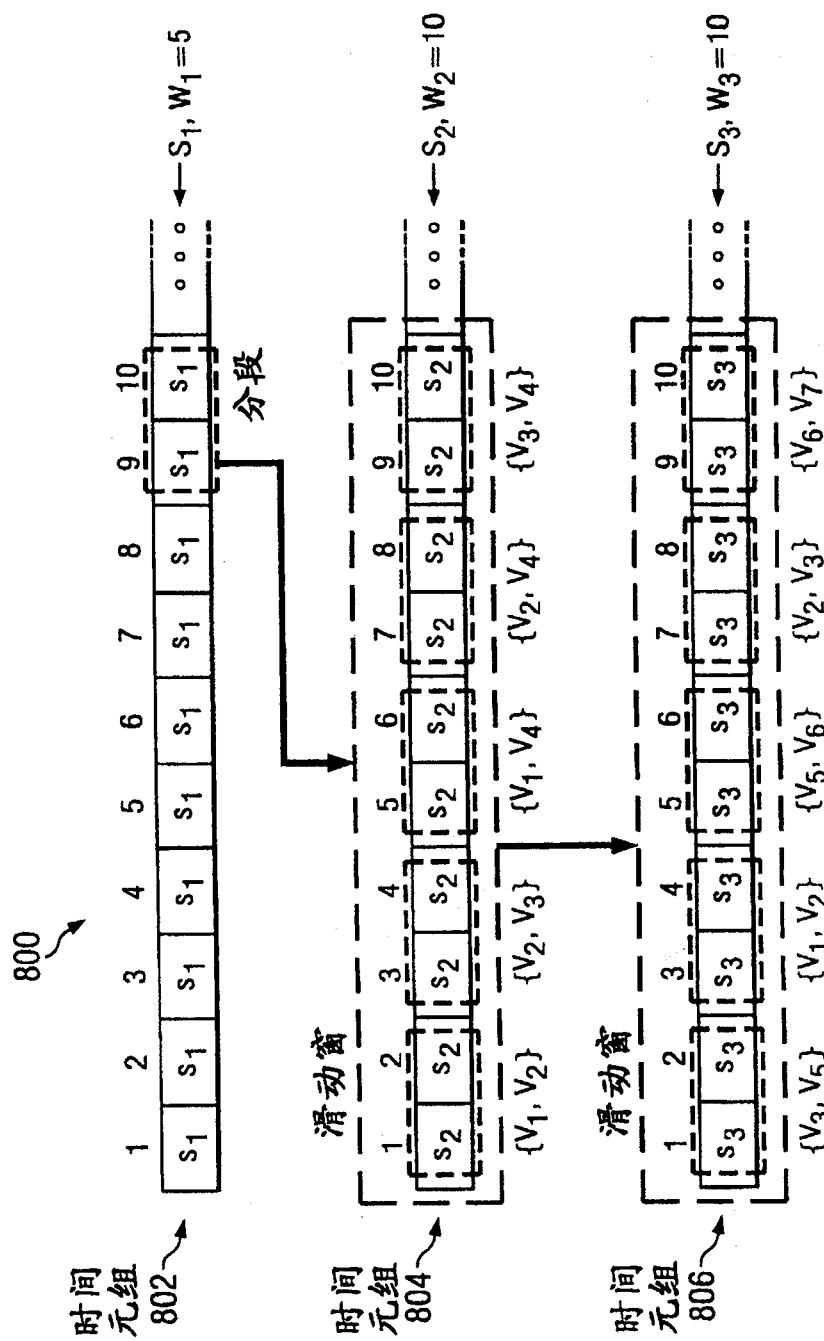


图 8

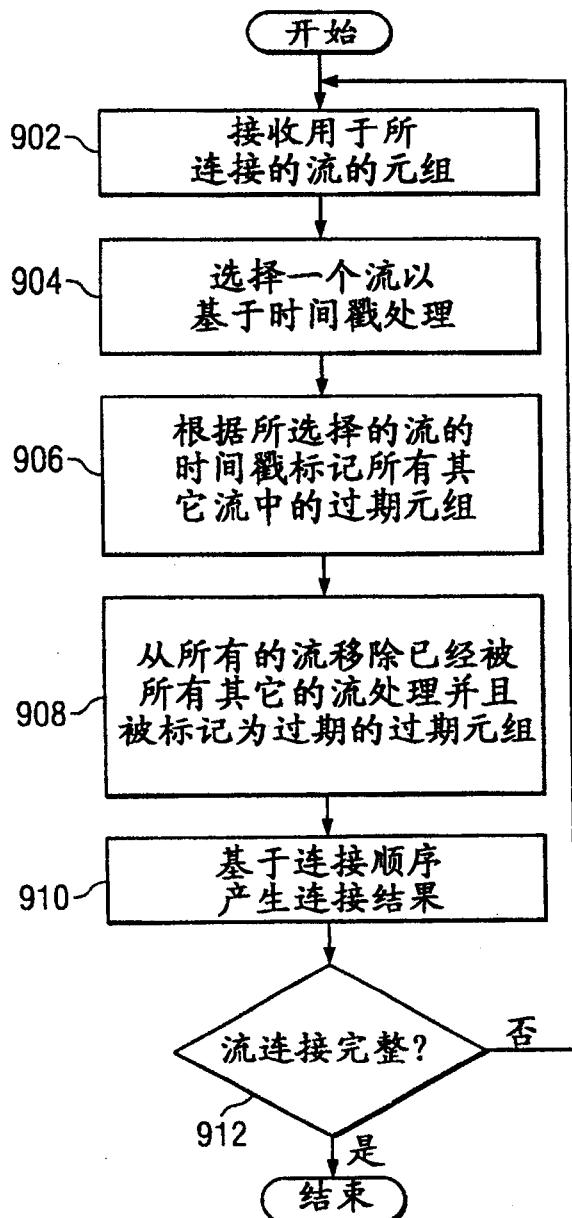


图 9

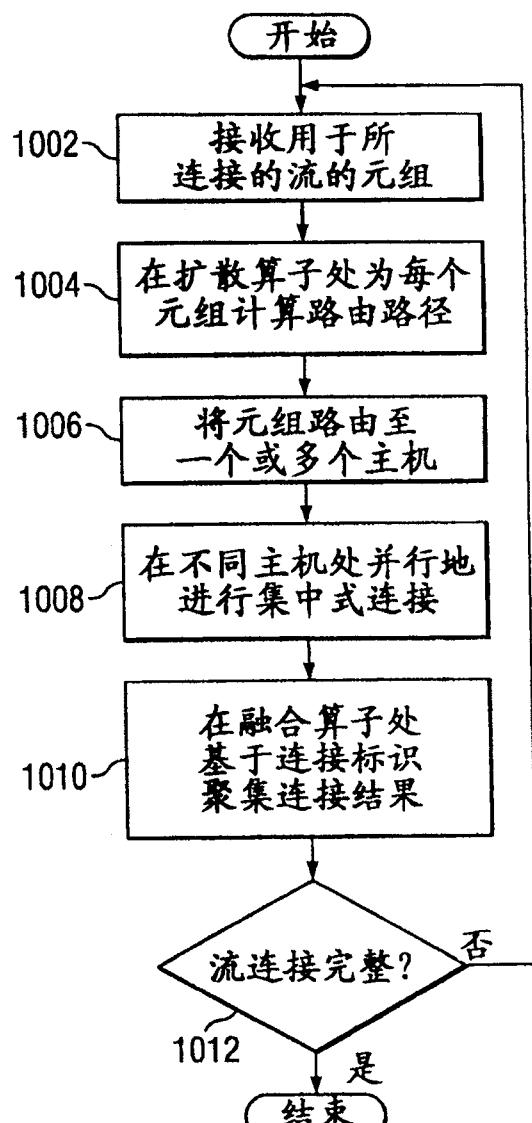
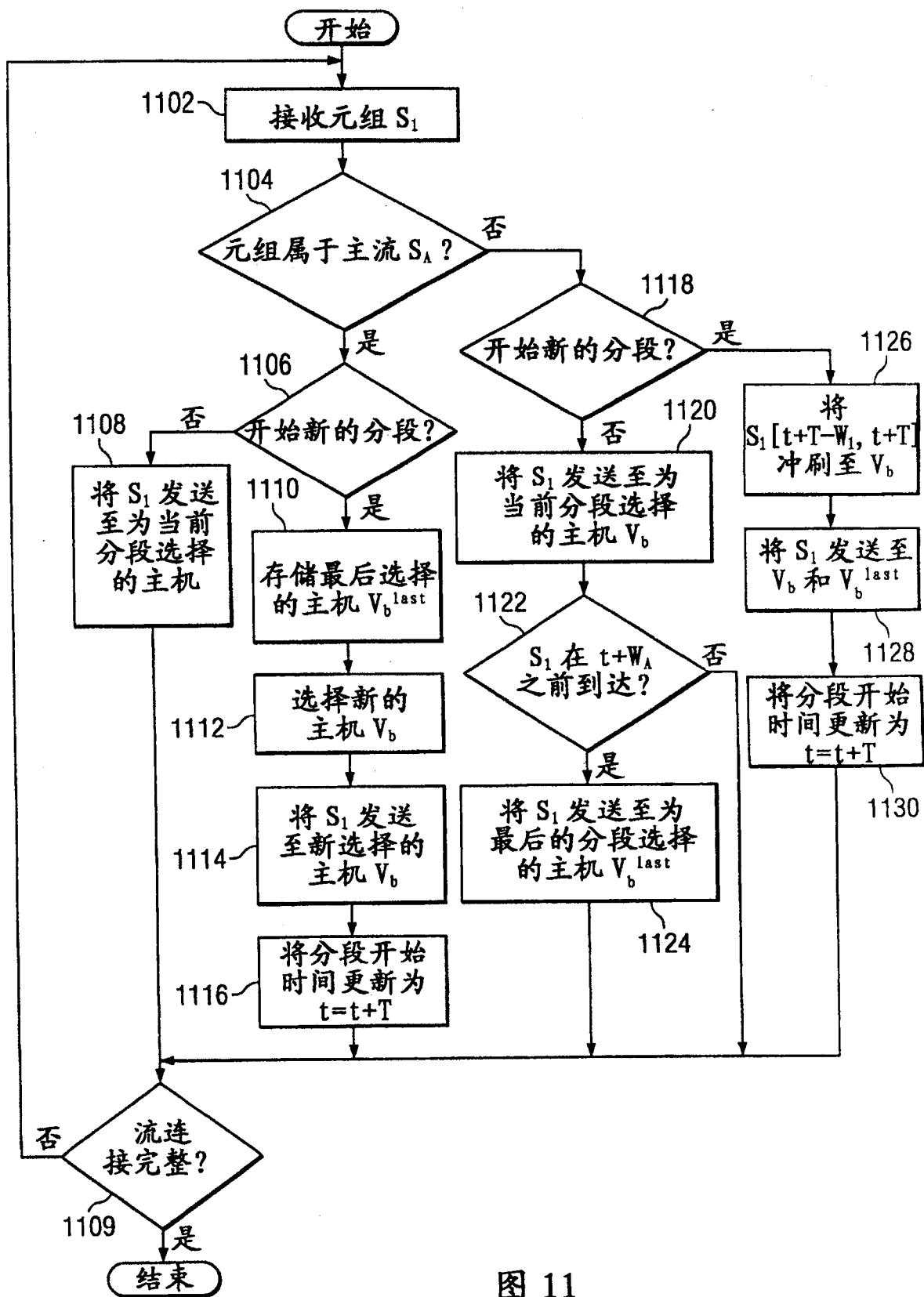


图 10



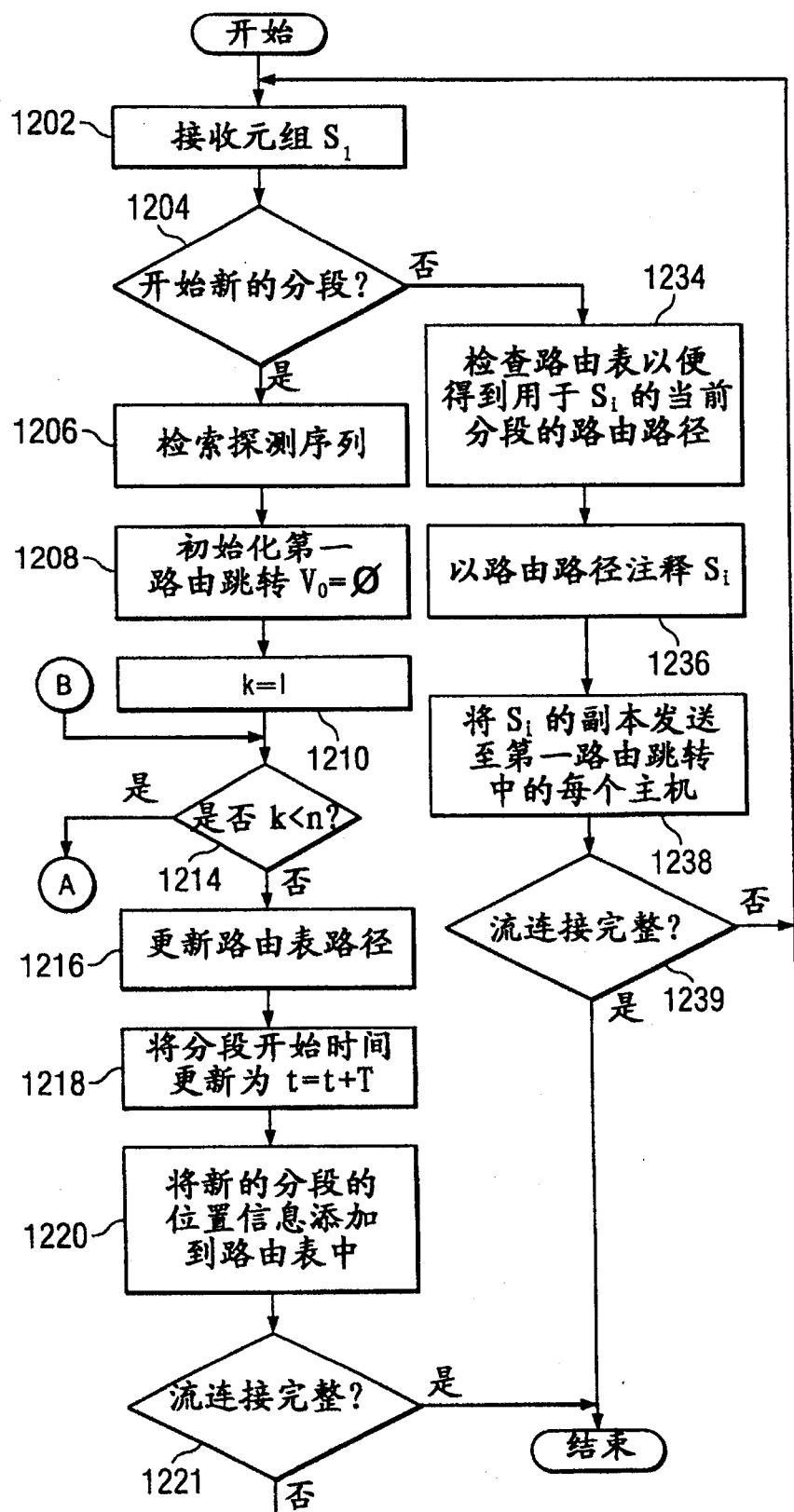


图 12A

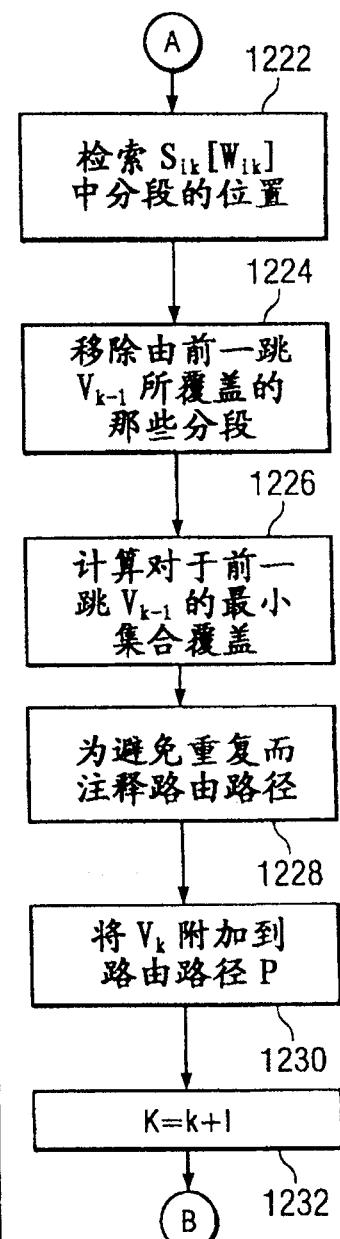


图 12B

1300 ↘

```

Procedure: Join ( $S_1[W_1], S_2[W_2], \dots, S_n[W_n]$ )
begin
1. while  $\exists Q_k$  that  $Q_k \neq \emptyset, 1 \leq k \leq n$ 
2. for  $s_k \in Q_k$  pointed by  $p_k, 1 \leq k \leq n$ 
3. select  $s_i$  such that  $s_i.t$  is the smallest
4. for all other queues  $Q_k, 1 \leq k \leq n, k \neq i$ 
5. mark all tuples  $\forall s_k \in Q_k, s_k.t < (s_i.t - W_k)$ 
6. remove  $\forall s_k$  processed and marked by all  $S_j, j \neq k$ 
7. get the probing sequence  $S_{i,1} \rightarrow \dots \rightarrow S_{i,n-1}$  for  $s_i$ 
8. set  $X_i = s_i$ 
9. for  $l = 1$  to  $n-1$ ,
10.  $X_i = X_i \bowtie S_{i,l}[W_{i,l}]$ 
11. produce join result  $X_i$ 
12. update  $p_l$  to point to the next tuple in  $Q_l$ 
end

```

图 13

1400 ↘

```

Procedure: ATR ( $S_1[W_1] \bowtie \dots \bowtie S_n[W_n], \{v_1, \dots, v_m\}$ )
begin
1. while receiving a tuple  $s_A \in S_A$ : master stream
2. If  $s_A.t \geq (t+T)$  /*start a new segment*/
3.  $v_b^{last} \leftarrow v_b$  and  $v_b$ : least-loaded host in  $\{v_1, \dots, v_m\}$ 
4. send  $s_A$  to  $v_b$ 
5.  $t \leftarrow t+T$  /*update the segment start time*/
6. else /* continue the current segment*/
7. send  $s_A$  to  $v_b$ 

8. while receiving a tuple  $s_i \in S_i, i \neq A$ : slave stream
9. If  $s_i.t \geq (t+T)$  /*start a new segment*/
10. flush  $S_i[t+T-W_i, t+T]$  to  $v_b$  /*flush  $S_i[W_i]$ */
11. send  $s_i$  to  $v_b$  and  $v_b^{last}$ 
12.  $t \leftarrow t+T$  /*update the segment start time*/
13. else /* continue the current segment*/
14. send  $s_i$  to  $v_b$ 
15. if  $s_i.t < (t+W_A)$ 
16. send  $s_i$  to  $v_b^{last}$  /*append  $S_i[W_A]$  to  $v_b^{last}$ */
end

```

图 14

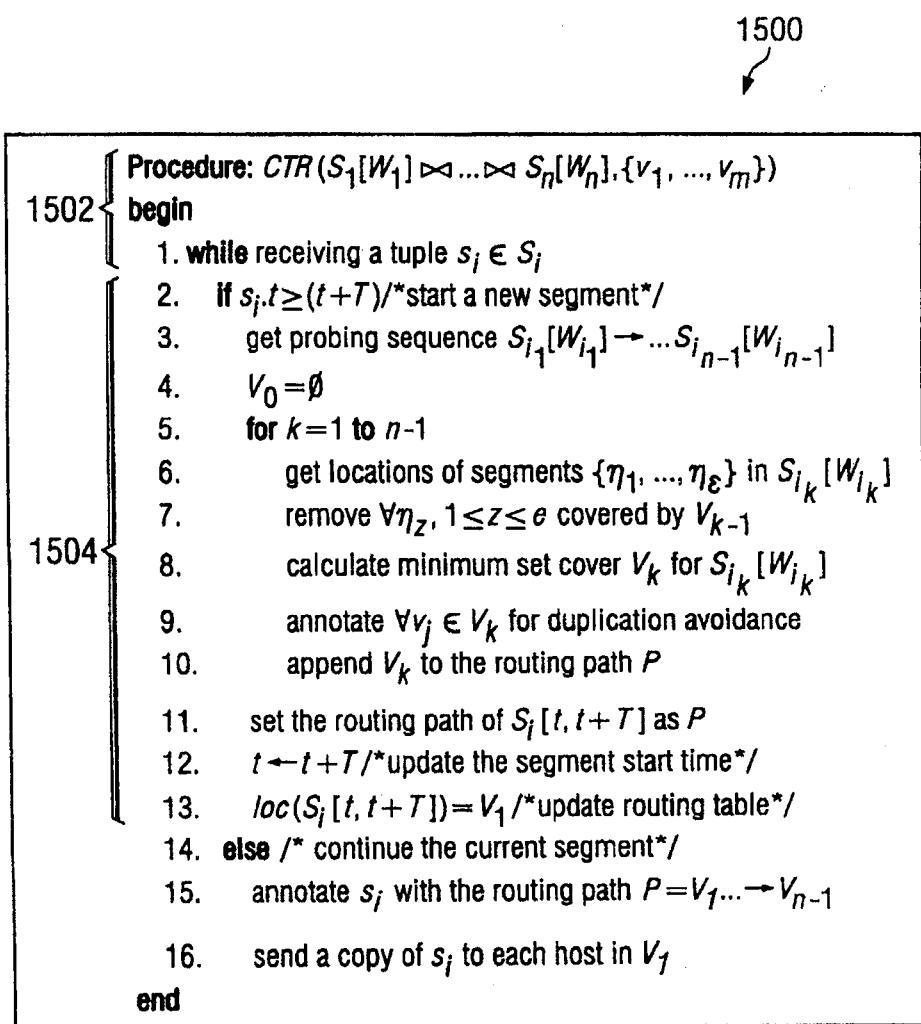


图 15