(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 101727481 A
(43)申请公布日 2010.06.09

(21)申请号 200910207939.4

(22)申请日 2009.11.02

(30)优先权数据
1869/KOL/2008 2008.10.31 IN

(71)申请人 软件股份公司
地址 德国达姆施塔特

(72)发明人 巴哈斯卡·雷迪·比雷迪
拉姆·拉玛阿 维内·普努斯

(74)专利代理机构 北京东方亿思知识产权代理有限责任公司 11258
代理人 李晓冬 南霆

(51)Int.Cl.
G06F 17/30(2006.01)
G06F 17/22(2006.01)

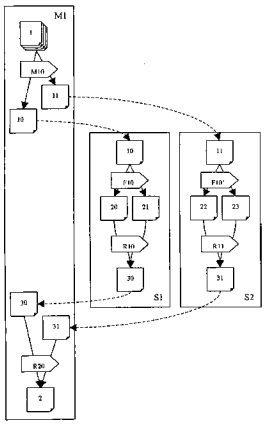
权利要求书 2 页 说明书 33 页 附图 5 页

(54)发明名称

对FLOW服务和大型文档进行映射化简的方法和服务器集群

(57)摘要

本发明提供了对FLOW服务和大型文档进行映射化简的方法和服务器集群。本发明涉及一种用于对电子数据交换(EDI)文档(1)的处理进行映射化简的方法，该方法包括以下步骤：a.将EDI文档(1)映射到多个中间文档(10,11)；b.处理中间文档(10,11)以产生多个中间结果(20-23)；c.化简多个中间结果(20-23)以产生多个经化简的中间结果(30,31)；以及d.化简经化简的中间结果(30,31)以产生表示经处理的EDI文档(1)的最终结果(2)。



CN 101727481 A

1. 一种用于对电子数据交换 EDI 文档 (1) 的处理进行映射化简的方法，该方法包括以下步骤：

a. 将所述 EDI 文档 (1) 映射到多个中间文档 (10,11)；

b. 处理所述中间文档 (10,11) 以产生多个中间结果 (20-23)；

c. 化简所述多个中间结果 (20-23) 以产生多个经化简的中间结果 (30,31)；以及

d. 化简所述经化简的中间结果 (30,31) 以产生表示经处理的 EDI 文档 (1) 的最终结果 (2)。

2. 如权利要求 1 所述的方法，其中，所述 EDI 文档 (1) 是以下述方式来映射的：使得所述中间文档 (10,11) 中的每一个包括所述 EDI 文档 (1) 的多个交换包封中的至少一个、多个功能组包封中的至少一个和 / 或多个事务集包封中的至少一个。

3. 如权利要求 1 或 2 所述的方法，其中，步骤 a. 和 d. 是由服务器集群的主服务器 (M1) 执行的，并且步骤 b. 和 c. 是由所述服务器集群的多个从属服务器 (S1，S2) 执行的，每个从属服务器 (S1,S2) 处理一个或多个中间文档 (10,11) 并且化简一个或多个中间结果 (20-23)。

4. 如权利要求 3 所述的方法，还包括以下步骤：通过异步调用将所述中间文档 (10,11) 从所述主服务器 (M1) 发送到所述从属服务器 (S1，S2)。

5. 如权利要求 3 所述的方法，其中，所述 EDI 文档 (1) 被存储在所述从属服务器 (S1，S2) 可访问的分布式文件系统中，并且所述方法还包括以下步骤：通过异步调用把对所述中间文档 (10,11) 的引用标记发送到所述从属服务器 (S1，S2)。

6. 如前述权利要求中任何一项所述的方法，其中，所述中间结果 (20-23) 中的每一个包括把相应中间结果 (20-23) 与所述 EDI 文档 (1) 联系起来的标识符。

7. 如前述权利要求中任何一项所述的方法，其中，用于执行步骤 b. 中所述从属服务器 (S1，S2) 的处理的处理逻辑是在运行时期间被分发到所述从属服务器 (S1，S2) 的。

8. 一种服务器集群，包括适于执行如权利要求 1-7 中任何一项所述的方法的主服务器 (M1) 和多个从属服务器 (S1，S2)。

9. 一种用于对 FLOW 服务的至少一个输入 (1) 的处理进行映射化简的方法，该方法包括以下步骤：

a. 通过映射器服务 (M10) 将所述 FLOW 服务的至少一个输入 (1) 映射到多个中间输入 (10,11)；

b. 执行所述 FLOW 服务的多个实例 (F10,F10')，所述 FLOW 服务的实例 (F10,F10') 处理所述中间输入 (10,11) 以产生多个中间结果 (20-23)；

c. 通过多个第一化简器服务 (R10，R11) 将所述中间结果 (20-23) 化简为多个经化简的中间结果 (30,31)；以及

d. 通过第二化简器服务 (R20) 来化简所述经化简的中间结果 (30,31) 以从所述经化简的中间结果 (30,31) 产生所述 FLOW 服务的最终输出 (2)。

10. 如权利要求 9 所述的方法，其中，所述映射器服务 (M10) 和所述第二化简器服务 (R20) 是在服务器集群的主服务器 (M1) 上执行的，并且所述 FLOW 服务的多个实例 (F10，F10') 和所述多个第一化简器服务 (R10,R11) 是在所述服务器集群的多个从属服务器 (S1，S2) 上执行的。

11. 如权利要求 9 或 10 所述的方法，其中，所述映射器服务 (M10) 的输入签名符合所述 FLOW 服务的输入签名。

12. 如权利要求 9-11 所述的方法，其中，所述化简器服务 (R10，R11，R12) 的输出签名符合所述 FLOW 服务的输出签名。

13. 如权利要求 9-12 所述的方法，其中，所述 FLOW 服务的至少一个输入包括电子数据交换 EDI 文档 (1)。

14. 一种服务器集群，包括适于执行如权利要求 9-13 中任何一项所述的方法的主服务器 (M1) 和多个从属服务器 (S1，S2)。

15. 一种计算机程序，包括适于实现如前述权利要求 1-7 或 9-13 中任何一项所述的方法的指令。

对 FLOW 服务和大型文档进行映射化简的方法和服务器集群

技术领域

[0001] 本发明涉及用于对例如电子数据交换 (EDI) 文档这样的大型文档的处理进行映射化简 (MapReduce) 的方法、服务器集群和计算机程序。

背景技术

[0002] 企业环境中的现代软件应用通常被组织成多个子程序，其中每个子程序执行该软件应用的某些子任务。通常，例如在不同企业的应用之间的通信领域中，这种应用必须处理巨大量的数据，其中必须发送和处理大型文档。

[0003] 这种应用通常是在集成服务器上执行的，集成服务器的一个示例是申请人的 webMethods 集成服务器。该集成服务器支持一种图形编程模型 FLOW，FLOW 用于定义集成服务器的处理逻辑。FLOW 允许将多个 FLOW 服务图形化地定义为“黑盒”服务以及这些 FLOW 服务之间的管道，这些管道用于将数据从一个 FLOW 服务的输出传递到另一 FLOW 服务的输入。由于 FLOW 是图形编程语言，因此它减轻了开发者编写复杂且易出错的传统代码的负担。FLOW 服务可用于处理任何种类的信息并用于执行各种计算。

[0004] 现有技术已知的一种用集成服务器来处理大型文档的常见方法是以顺序方式处理文档的内容。然而，由于文档的大小可能在千兆字节的范围内，因此这种顺序处理是非常耗时且处理密集的，并且可能要求特殊的高端硬件，而这种硬件的维护是成本高昂且复杂的。

[0005] 另一种现有技术已知的方法是采用代理 (broker)，该代理将大型文档分发到集成服务器的多个实例，以便实现某种并行处理。然而，该方法要求额外的并且经常是复杂的消耗传递中间件，以便在代理和集成服务器实例之间进行通信，这通常会造成较高的网络带宽要求并且导致对资源的高消耗。另外，该方法通常涉及由代理和集成服务器实例处理多个大型文档，其中每个大型文档仍由单个集成服务器以顺序方式来处理。

[0006] 另外，在处理大型输入数据集的领域中，从 Google 公司的 J.Dean 等人所著的文献“MapReduce :Simplified Data Processing on Large Clusters”(OSDI ′ 04 : Sixth Symposium on Operating System Design and Implementation, San Francisco, December,2004) 中知道了一种编程模型和相关联的框架，其被称为 MapReduce(映射化简)。用户定义的映射 (map) 函数取得输入的对，并产生中间键 / 值对的一个集合。MapReduce 库把与同一中间键相关联的所有中间值集合在一起，并把它们传递到用户定义的化简 (reduce) 函数。化简函数接受一中间键和一个值集合。它将这些值合并在一起以形成一个可能较小的值集合。通常每次化简调用将产生零个或一个输出值。中间值经由迭代器被提供到用户的化简函数。这样就允许了处理太大以至于不能装入存储器中的值列表。以这种编程模型编写的程序可以自动被该框架在不同机器上并行执行。然而，将 MapReduce 编程模型用于现有应用上要求对该应用的编程逻辑进行深入的适应性修改以便符合 MapReduce 编程模型。另外，MapReduce 是意图用于搜索引擎领域的，在这个领域中，

诸如对大批文档中的单词计数、构建 web 链接的图结构等等之类的专门任务是常见的。

[0007] 大型文档处理的一个具体示例是电子数据交换 (EDI)。EDI 涉及通过电子手段在应用之间传输结构化的消息。EDI 通常用于在不同企业的应用之间传输诸如发票或购买定单之类的大型文档。结构化消息的若干种标准化结构是现有技术中已知的，例如 ANSI X12、UCS、VICS、UN/EDIFACT、ODETTE 和 EANCOM。对这种大型 EDI 文档的处理通常涉及上述的缺点。

[0008] 因此，本发明所基于的技术问题一方面是提供一种方法和系统，用于以较少的处理时间和计算工作来处理大型文档尤其是 EDI 文档，从而至少部分地克服以上说明的现有技术的缺点。本发明所基于的另一个相关的技术问题是提供一种方法和系统，用于以较少的处理时间和计算工作来处理 FLOW 服务的输入，并且其只需要最低限度的适当性修改工作就可灵活地适应于现有的编程逻辑。

发明内容

[0009] 根据一个方面，本发明涉及一种用于对电子数据交换 (EDI) 文档的处理进行映射化简的方法。在权利要求 1 的实施例中，该方法包括以下步骤：

[0010] a. 将 EDI 文档映射到多个中间文档；

[0011] b. 处理中间文档以产生多个中间结果；

[0012] c. 化简多个中间结果以产生多个经化简的中间结果；以及

[0013] d. 化简经化简的中间结果以产生表示经处理的 EDI 文档的最终结果。

[0014] 本发明的第一方面是基于以下认识的，即 MapReduce 的概念不仅可用在搜索引擎的上下文中，而且也可有利地用于企业环境中的 EDI 文档的处理。因此，大型 EDI 文档首先被映射即分割为多个中间文档。映射即分割优选是这样执行的：使得每个所得到的中间文档具有大致相等大小的有效载荷，即，使得当其在该方法的其他步骤中处理时消耗同等量的处理时间和 / 或处理资源。

[0015] 中间文档随后被处理以产生多个中间结果，这优选是并行执行的，以便改善就整体处理时间而言的处理性能。另外，由于 EDI 文档被映射到多个通常更小的中间文档，因此中间文档可通过普通硬件来处理，即不需要采用专门的高端硬件。

[0016] 在对中间文档进行处理之后，所得到的中间结果被化简以产生多个经化简的中间结果。化简意味着将相关的中间结果整合为一个经化简的中间结果。此上下文中的“相关”指的是两个或更多个中间结果源自同一原始 EDI 文档。

[0017] 最后，经化简的中间结果在另一个步骤中被化简以产生最终结果。该方法步骤通常涉及适当地组合经化简的中间结果，以便获得 EDI 文档处理的最终结果。

[0018] 如果化简步骤是在不同物理机器上执行的以便实现并行化，则上述的两步化简尤其有利。在此情况下，由于中间结果在被发送到执行第二化简步骤的另一机器之前已被化简，因此宝贵的网络带宽可得以节省，因为在机器之间需要传输的结果更少。另外，如果化简步骤是满足交换律 (即，A operation B 相当于 B operation A) 和 / 或满足结合律 (即，A operation(Boperation C) 相当于 (A operation B)operation C) 的操作，则这个方面尤其有利。因此，化简步骤可以按任何顺序并行执行。与两步化简相关联的另一个优点在于负担即用于执行化简步骤的处理时间可被分摊在多个普通机器上，而不是一个机器在一个

大集合上进行化简步骤。第二化简步骤从而可在较小的中间结果集合上执行。

[0019] 在本发明的另一个方面中，EDI 文档 (1) 可以按下述方式来映射：使得中间文档 (10,11) 中的每一个包括 EDI 文档 (1) 的多个交换包封中的至少一个、多个功能组包封中的至少一个和 / 或多个事务集包封中的至少一个。因此，映射即分割可在由 EDI 文档的结构定义的边界之一处执行，即在事务集包封级上执行、在功能组包封级上执行和 / 或在交换包封级上执行。它通常依赖于最终用户用来基于 EDI 文档的结构定义在哪个边界上分割 EDI 文档。例如，如果功能组和 / 或交换包封包含最优数目的事务来定义合理大小的有效载荷。

[0020] 步骤 a. 和 d.，即映射和最终化简，可由服务器集群的主服务器执行，并且步骤 b. 和 c.，即分别处理和化简中间文档或中间结果，可由该服务器集群的多个从属服务器执行。每个从属服务器可处理一个或多个中间文档并且化简一个或多个中间结果。通过多个从属服务器来执行处理是尤其有利的，因为中间文档的处理能够被高度并行化。主服务器和从属服务器优选是通过网络连接通信的不同物理机器。

[0021] 例如，如果处理任务是将一列数字 {1,2,3,4,5,6,7,8,9,10,11,12} 加起来，则主服务器可将中间文档 {1,2} 委托给从属节点 1，将 {3,4} 委托给从属节点 2，将 {5,6} 委托给从属节点 3，将 {7,8} 委托给从属节点 1，将 {9,10} 委托给从属节点 2，并且将 {11,12} 委托给从属节点 3。在从属节点 1 处，中间结果于是将是中间文档的加和：对应于 {1,2} 的 3，以及对应于 {7,8} 的 15。从属节点 1 上的化简步骤于是将把 3 和 15 相加，得到 18。相应地，从属节点 2 上的化简步骤将把 7 和 19 相加以得出 26，并且从属节点 3 将把 11 和 23 相加以得到 34。因此，只有三个经化简的中间结果 18、26、34 需要被传送回主服务器，而不是传送 3、15、7、19、11、13。在主服务器上执行的最终化简步骤于是将得出 78(18+26+34)，这是期望的结果。

[0022] 在另一个方面中，该方法还包括以下步骤：通过异步调用将中间文档从主服务器发送到从属服务器。因此，主服务器取得大型 EDI 文档并将中间文档的处理委托给从属服务器。EDI 文档本身优选地与主服务器保持在一起。异步调用意味着一旦主服务器调用即触发了从属服务器的处理（这优选由主服务器的线程池来执行），主服务器线程并不等待从属服务器完成其处理（这将是同步调用），而是主服务器可以立即继续进行其自己的处理，即随后调用其他从属服务器。此概念进一步增大了本发明的处理速度，因为不存在被阻塞的主服务器资源（即，等待从属服务器），从而使得能够将任务更快地委托给从属服务器。

[0023] 或者，EDI 文档可被存储在从属服务器可访问的分布式文件系统中，并且该方法还可包括以下步骤：主服务器通过异步调用把对中间文档 (10,11) 的引用标记 (reference) 发送到从属服务器。如果从属服务器通过直接连接与该分布式文件系统相连接，则这个方面可以大大加快处理，因为不必通过缓慢的网络连接来发送 EDI 文档或中间文档。在此情况下，只有对 EDI 文档和 / 或 EDI 文档的应当被从属服务器处理的部分（即，中间文档）的引用标记需要被传递到从属节点。协同定位 (co-location)（即，提供对实际存在于从属节点本身上的部分的引用标记）尤其有利，因为它节省了许多带宽消耗，因为在机器之间不发生 EDI 数据传送。

[0024] 当处理中间文档时，从属服务器优选地在本地将中间结果存储在存储器或持续性文件系统中，这些中间结果随后被主服务器所收集。

[0025] 另外，中间结果中的每一个可包括把相应中间结果与 EDI 文档联系起来的标识符。通过使用标识符，这些中间调用结果中的每一个可被追踪到原始调用。标识符可以是例如随着利用大型 EDI 文档进行的每次原始调用而增大的计数器。标识符可用于允许在主服务器调用从属服务器时的异步行为。这个方面可以释放主服务器处的委托线程（在同步模式中，这些线程必须等待从属服务器执行其处理），从而使得主服务器处的资源利用更好并且间接地带来了更大的并行化。当主服务器以异步方式将中间结果委托给从属服务器时，从属服务器从而拥有了将其获得的中间结果追踪回来自主服务器的原始调用的手段。例如，如果存在要处理的大型 EDI 文档，则可为该调用创建标识符“12345”。该方法可在将中间文档委托给从属服务器的同时将该标识符传递到从属服务器。这帮助了在后续化简步骤中将所有中间结果与原始 EDI 文档联系起来，因为在从属服务器处可利用此标识符来维护中间结果。

[0026] 作为附加或替换，用于执行步骤 b. 中从属服务器的处理的处理逻辑可在运行时期间被分发到从属服务器。因此，从属服务器不必拥有执行 EDI 文档的可执行文件即处理逻辑的拷贝。这些可执行文件例如可被包括在主服务器的库中并在运行时被散布到从属服务器。散布优选地是依据当前 EDI 文档所需的可执行文件来执行的。任何对等框架或专属机制都可用于共享可执行文件。

[0027] 另外，本发明涉及一种服务器集群，其包括适合用于执行以上给出的方法中的任何一种的主服务器和多个从属服务器。

[0028] 在本发明的另一个方面中，提供了一种用于对 FLOW 服务的至少一个输入的处理进行映射化简的方法。在权利要求 9 的实施例中，该方法包括以下步骤：

[0029] a. 通过映射器服务将 FLOW 服务的至少一个输入映射到多个中间输入；

[0030] b. 执行 FLOW 服务的多个实例 (F10,F10')，FLOW 服务的实例处理中间输入以产生多个中间结果；

[0031] c. 通过多个第一化简器服务将中间结果化简为多个经化简的中间结果；以及

[0032] d. 通过第二化简器服务来化简经化简的中间结果以从经化简的中间结果产生 FLOW 服务的最终输出。

[0033] 因此，FLOW 服务（不论是新创建的还是现有的 FLOW 服务），并不以顺序方式来处理其输入，而是 FLOW 服务的处理通过上述方法被有效地“并行化”。因此，FLOW 服务的输入不像通常执行的那样被直接馈送到 FLOW 服务中，而是首先被映射器服务分割成多个中间输入。在一个实施例中，FLOW 服务本身被“克隆”，即中间输入被 FLOW 服务的多个实例所处理，优选是并行地处理。所得到的中间结果随后被多个第一化简器服务所化简，以便对于 FLOW 服务的每个实例获得一个经化简的中间结果。最后，经化简的中间结果被第二化简器服务所化简，以便提供 FLOW 服务的最终输出。优选地，第二化简器服务是基于与多个第一化简器服务相同的实现的，即，所有化简步骤都是由同一化简器服务的多个实例执行的。在下文中，为清楚起见，所使用的术语“化简器服务”和“化简器服务的实例”是同义的。应当注意，FLOW 服务的整体输入和输出保持相同，只不过处理被并行化了。

[0034] 在一个方面中，映射器服务和第二化简器服务是在服务器集群的主服务器上执行的，并且 FLOW 服务的多个实例和多个第一化简器服务是在所述服务器集群的多个从属服务器上执行的。

[0035] 在另一个方面中，映射器服务的输入签名符合 FLOW 服务的输入签名。作为附加或替换，化简器服务的输出签名符合 FLOW 服务的输出签名。输入签名（或输出签名）优选地定义作为服务的输入（或输出）提供的变量的数目和类型，因此定义了服务的接口。

[0036] 由于映射器服务的输入签名优选地符合要并行化的 FLOW 服务的输入签名这一事实，因此任何现有的 FLOW 服务都可被连接到具有相同输入签名的映射器服务。另外，任何现有的 FLOW 服务都可被连接到具有相符的输出签名的化简器服务，这意味着任何现有的 FLOW 服务都可被嵌入在本方法中，而无需对其输入或输出签名或内部处理逻辑进行适应性修改。这是尤其有利的，因为它大大增加了本发明的灵活性和可应用性。输入和输出签名的示例在以下详细描述中给出。

[0037] 在另一个方面中，FLOW 服务的至少一个输入可包括电子数据交换 (EDI) 文档。因此，FLOW 服务在这个方面中优选适合用于处理 EDI 文档。当 FLOW 服务被并行化时，可以实现 EDI 文档的尤其高效的处理，这类似于以上进一步给出的那些方面。然而，应当明白，FLOW 服务完全不限于处理 EDI 文档。相反，FLOW 服务适合于处理任何种类的文档，例如 XML 文档。另外，不仅可通过 FLOW 服务处理文档，还可实现任何种类的计算逻辑。

[0038] 本发明还涉及一种服务器集群，其包括适合用于执行以上给出的方法中的任何一种的主服务器和多个从属服务器。

[0039] 最后，提供了一种计算机程序，其包括适合用于实现上述方法中的任何一种的指令。

附图说明

[0040] 在以下详细描述中，参考以下附图进一步描述本发明的当前优选的实施例：

[0041] 图 1：本发明的实施例的示意性概览；

[0042] 图 2：根据本发明实施例的主服务器和多个从属服务器的示意性概览；

[0043] 图 3：EDI 文档的结构概览；

[0044] 图 4：示例性 FLOW 服务及其相关输入和输出；

[0045] 图 5：另一个用于处理 EDI 文档的示例性 FLOW 服务；

[0046] 图 6：用于指定被映射化简的 FLOW 服务的属性的图形用户界面的概览；以及

[0047] 图 7：本发明的实施例的示例性实现方式的类图。

具体实施方式

[0048] 在下文中，针对服务器集群根据本发明对大型 EDI 文档的处理来描述本发明的当前优选的实施例。如图 2 中示意性示出的，也被称为网格的服务器集群是一种分布式的计算平台，其允许并行处理。它通常包括联网的、松散耦合的计算机的集群，这些计算机协同动作以执行极大的计算或数据密集型任务。应当明白，处理 EDI 文档仅是本发明的许多种场景中的一种，任何其他类型的文档也可被处理。另外，本发明不仅可以有利地实现文档处理，还可以有利地实现任何种类的复杂计算，以下的更多示例性实施例将证实这一点。

[0049] EDI 文档的一般结构在图 3 中示意性地示出，图 3 示出了例如由 ANSIASC X12 标准定义的结构。因此，EDI 文档包括任意数目的事务，这些事务被按各种包封 (envelop) 来分组。在最内部的级别上，事务集由图 3 所示的 ST/SE 片段 (segment) 来标识。ST 片段优选

地包括事务集 ID、控制号码和可选的实现规约参考标记。SE 片段优选地包括该事务集中包括的片段的数目以及与 ST 片段相同的控制号码。第二包封级别是功能组包封。其目的是对一次传输内的相似类型的事务集分组。ANSI ASC X12 定义了若干个用于对类似的事务集分组的业务过程，比如规划调度 (830)、购买定单 (850)、购买定单确认 (855)、购买定单改变 (865)、定单状态查询 (869) 或定单状态报告 (870)。

[0050] 最外部的级别是由 ISA 和 IEA 片段定义的交换包封（参见图 3）。交换包封优选地包含了从一个发送者到一个接收者的数据。ISA 片段优选为固定长度的片段。ISA/IEA 片段中包含的一些项目是发送者和接收者的结构化的邮箱地址、交换控制号码、交换包封内的功能组计数、时间 / 日期戳以及交换包封的版本。

[0051] 处理这种 EDI 文档的通常方式可能是将 EDI 文档的数据映射到另外的格式（例如，后端系统所要求的格式），或者将来自 EDI 文档的数据映射到 FLOW 服务的输入，下文将进一步概述这一点。

[0052] 传统的 EDI 处理通常一次处理一个事务。如果 EDI 文档大小大约为数百兆字节或千兆字节，则此处理是非常耗时的。为了在某种程度上减轻这一缺点，通常部署高端服务器的集群来并行地处理多个 EDI 文档中的每一个。然而，对高端服务器的采用具有严重的缺点，例如，如果在处理期间硬件 / 软件发生故障则复杂度会增大，以及所有者为了维护高端服务器而成本增加。

[0053] 本发明定义了一种用于在 EDI 文档级上并行化处理的方法和服务器集群。从图 1 可以看出，主服务器 M1 首先接收大型 EDI 文档 1。该 EDI 文档首先在交换包封边界上被映射即分割成多个中间文档 10、11。然而，应当明白，取决于 EDI 文档的类型，在本发明的其他实施例中 EDI 文档也可在例如功能组包封级或者甚至在事务集包封级被分割。

[0054] 甚至更细粒度的方法也适合于本发明，例如在单事务级分割 EDI 文档，如果 EDI 文档中的事务是独立的实体的话。结果，文档可被映射（分块）为非常小的部分，从而带来了高水平的并行化。

[0055] 在分割 EDI 文档之后，主服务器 M1 将中间文档 10、11 委托给多个从属服务器 S1、S2 处理。从属服务器 S1、S2 处理中间文档 10、11 并且产生中间结果 20-23。应当注意，对一个中间文档的每次处理可能产生多个中间结果，下文将进一步说明这一点。

[0056] 中间结果 20-23 随后被从属服务器 S1、S2 中的每一个化简，以便优选地在每个从属服务器 S1、S2 中获得一个经化简的中间结果 30、31。

[0057] 当主服务器 M1 完成了将中间文档委托给从属服务器 S1、S2 时，它优选地在从属服务器 S1、S2 中的每一个上发出化简调用。委托优选地是以异步方式调用的，从而使得主服务器 M1（即，其线程）可以继续进行其处理而不必等待每个从属服务器 S1、S2 完成执行，这一点已在上文中说明了。

[0058] 主服务器 M1 发出的化简调用引起从属服务器 S1、S2 将其各自化简的中间结果 30、31 发送回主服务器 M1。主服务器 M1 随后发出另一化简调用以便将收集到的经化简的中间结果 30、31 整合为一个最终输出 2。输出 2 于是代表了经处理的 EDI 文档 1。

[0059] 应当注意，由于从属服务器 S1、S2 中的每一个仅处理整个 EDI 文档 1 的一部分，因此不需要专门的高端硬件。普通设备可用作从属服务器，这大大降低了整个体系结构的成本。

[0060] 主服务器和从属服务器的处理优选的由若干个服务来执行。尤其优选的是其中服务器是 webMethods 集成服务器的实施例。webMethods 集成服务器处于申请人的 webMethods 系列产品的核心处。它是基于 Java 的多平台企业集成引擎，该引擎支持服务的执行，以执行诸如数据映射和与其他系统的通信之类的集成逻辑。该集成服务器提供了一种图形编程模型 FLOW，FLOW 用于执行诸如映射、调用其他服务、循环和分支之类的常见集成任务。该集成服务器的一些特征包括编写图形 FLOW 和 java 服务，定义和修改文档和映射逻辑，测试、调试和执行服务，创建和配置 web 服务，以及编辑适配器服务和通知。

[0061] 图 4 示出了示例性的简单 FLOW 服务“sampleFlowService”，其取得两个整数“input1”和“input2”，并且提供两个输出“multiplyIntsResult”（将两个输入整数相乘的结果）和“addIntsResult”（将两个输入整数相加的结果）。当在集成服务器上执行该示例性 FLOW 服务时，可向用户提供用于输入这些输入的值的对话，并且可呈现包括计算结果的另一对话。图 4 示出了被开发者优选用于指定输入、FLOW 服务和输出之间的映射的图形用户界面。

[0062] FLOW 服务处理的另一示例是对文件中单词的出现计数。常见的没有并行化的方法是逐行地读取该文件，将单词作为键添加在 HashMap 中并将计数作为值添加在 HashMap 中。首先，就该键来查询 HashMap，并且如果查询返回“null”（空），则计数被置为 1。否则，原始计数被取得，并且它将被递增并被放回 HashMap 中。当映射器 M10 和化简器服务 R10、R11、R20 被编写时，映射器服务可以产生较小的文件作为输出，并且化简器服务仅将输出 HashMap 与最终 HashMap 相组合。因此，进行单词计数的原始 FLOW 服务的输入 / 输出签名保持相同，而只有映射器和化简器操作的逻辑需要被编写。这是本发明的一个尤为有利的方面，下文将进一步说明这一点。

[0063] FLOW 服务的另一个示例是 EDI 文档的处理。图 5 示出了示例性的 FLOW 服务“mainService”，其取得 EDI 文件名作为输入。它通过调用服务“ediToValues”（也在图 5 中示出）来将 EDI 文件格式转换为内部 webMethods 格式。作为输出，它返回所输入的 EDI 文件在转换后是否整体上是有效的。它还可指示出执行服务所消耗的处理时间（图 5 中没有示出）。FLOW 服务“ediToValues”的输入 / 输出签名是如下组织的：它接受输入“ediFileName”(EDI 文档的文件名）或输入“edidata”（被表示为串的实际 EDI 数据本身），这两者是互斥的。如果“printTime”输入被设置，则执行服务所花的时间将被打印出来。单个输出“isValid”将被输出，以表明 EDI 文档在转换后是否有效。

[0064] 由于对上述 FLOW 服务“ediToValues”的处理顺序地消耗了大量处理时间和资源，因此以下论证本发明的方法如何被应用到这种现有的 FLOW 服务上以便高效并且灵活地对其进行“并行化”。

[0065] 参考图 6，在 FLOW 服务“ediToValues”的属性面板中，开发者可以提供以下属性：

[0066] • 映射器服务：用于映射输入数据的有效集成服务器服务

[0067] • 化简器服务：用于化简输出数据的有效集成服务器服务

[0068] • 网格使能：设置为“真”

[0069] • 扼流 (Throttle)：包括主服务器和从属服务器在内的并行执行的最大数目

[0070] • 策略：此属性指明是将从属服务器的中间结果保存在存储器中（如果它们的大小可忽略的话）还是将它们持续保留在文件中。

[0071] 然后本发明使用上述的映射器服务 M10(参见图 1)来执行将 EDI 文档 1 映射到多个中间文档 10、11。映射器服务 M10 的输入和输出签名优选地遵循某些规则：

[0072] • 映射器服务的输入优选地与被“并行化”的 FLOW 服务(在该示例中是“ediToValues”)相同。在此情况下，映射器服务接受与服务“ediToValues”的输入相匹配的输入“ediFileName”。

[0073] •映射器服务的输出优选地被包裹在名为“serviceInputData”的集成服务器文档中。“serviceInputData”的内容优选地是被“并行化”的 FLOW 服务的输入。在该示例中，映射器服务的输出“edidata”与服务“ediToValues”的输入相匹配。

[0074] •另外，映射器服务的输出优选地提供布尔型的“isLastSplit”。映射器服务在处理最后的映射步骤时将该值设置为“真”。映射器服务于是可反复地被调用，直到该值被设置为“真”为止。

[0075] 化简器服务 R10、R11、R20 的输入和输出签名优选地也遵循某些规则：

[0076] •化简器服务的输入被包裹在称为“reduceInputData”的集成服务器文档列表中。该文档列表优选地是文档阵列。该文档列表中的每个条目的内容可以是要被“并行化”的 FLOW 服务的输出。在该示例中，化简器服务的输入“isValid”与服务“ediToValues”的输出相匹配。

[0077] •化简器服务的输入还可提供布尔型的“isLastReduceStep”。如果化简器处理最后的化简调用，则该值可被设置为真。这可用于在化简器服务中执行清理活动。

[0078] • 化简器服务的输出应当是要被“并行化”的服务的输出。在该示例中，输出“isValid”与服务“ediToValues”的输出相匹配。

[0079] 可以看出，以上定义的映射器和化简器服务符合 FLOW 服务的输入和输出签名。这具有这样的优点，即任何现有的 FLOW 服务都可以很容易地被“并行化”，因为 FLOW 服务本身的签名和内部处理都不必被适应性修改。映射器和化简器服务只是简单地被分别“插入”在 FLOW 服务之前和之后。

[0080] 如果化简操作是满足结合律和交换律的，则以上给出的方法可被尤为有利地应用。例如，当计算 1 至 100 的范围中的质数的量时，可以采用两个输入分割；第一输入分割为 1 至 50，第二输入分割为 51 至 100。此示例中的中间输出将为分别表示这两个分割中的质数数目的“x”和“y”。化简操作将进行加法，该加法是可结合且可交换的。

[0081] 以上给出的签名符合性是本发明比现有技术已知的传统 MapReduce 算法有利的优点之一。用户编写的传统 MapReduce 的映射步骤取得输入的对并产生中间键 / 值对的集合，而根据本发明的集成服务器上的映射器服务遵循标准的签名并且只是对输入数据进行“分块”，即分割。另外，传统的 MapReduce 的映射步骤通常是在取得输入对并产生中间键 / 值对的集合的从属服务器上运行的，其中集成服务器上的映射器服务优选地在主服务器 M1 上执行，主服务器 M1 随后将分块的输入数据委托给从属服务器 S 1、S2 以便执行实际服务。这一点尤其灵活并且使得 flow 服务易于开发和维护，其若干原因如下：在传统的 MapReduce 算法中，不存在被“并行化”的服务，而其更确切地说是通过执行期望操作的映射器和化简器来定义的编程构造。与集成服务器中不同，不存在与期望操作相对应的服务。这使得所要求保护的方法易于理解并且尤其对用户友好。

[0082] 至于由用户编写的传统 MapReduce 的化简步骤，它取得中间键并且该键的值集

合，并且将这些值合并以形成可能较小的值集合。相反，当化简器服务在根据本发明的集成服务器上执行时，主服务器 M1 优选地向所有从属服务器 S1、S2 发出化简调用，以整合相关的中间结果。当从属服务器 S1、S2 中的每一个中的化简操作之后主服务器 M1 从从属服务器 S1、S2 取回结果时，主服务器 M1 在内部将这些结果组合成一个最终结果。这实质上使得化简操作成为一个两步过程，该过程首先在从属服务器 S1、S2 上执行，然后在主服务器 M1 上执行，这节省了网络带宽，从而带来了处理时间的进一步减少以及对资源的更好利用，以上已对这一点进行了说明。

[0083] 本发明的服务器集群的其他特征也是可能的。主集成服务器例如可维护一配置文件，该配置文件包括可用的从属服务器的列表。它可包括主服务器将处理委托给从属服务器所需要的信息。这个简单的工具很容易被扩展来实现从属节点的动态识别。例如，当从属服务器启动时，它可以向服务器集群中的所有机器广播其标识，并且主服务器可以将该从属服务器识别为潜在的从属服务器。

[0084] 在下文中，给出本发明的实施例的示例性 Java 实现，其主要组件在图 7 中示出。然而，应当明白，本发明既不限于编程语言 Java，也不限于下文中示出的具体实现。

[0085] 图 7 所示的类 JobClient 用于定义“作业”(job)，该作业代表根据本发明对数据处理的一次执行。JobClient 的示例性实现在以下代码列表中示出：

[0086]
```
package com.wm.app.b2b.server.mapred;
```

[0087]
```
import com.wm.data.IData;
```

[0088]
```
import com.wm.lang.ns.NSName;
```

[0089]
```
import com.wm.util.UUID;
```

[0090]
```
public class JobClient
```

[0091]
```
{
```

[0092]
```
        private JobConfiguration jobConf = null;
```

[0093]
```
        private JobInProgress jobInProgress = null;
```

[0094]
```
        protected NSName mapper = null;
```

[0095]
```
        protected NSName reducer = null;
```

[0096]
```
        protected NSName mapTaskName = null;
```

[0097]
```
        protected int throttle = 0;
```

[0098]
```
        protected boolean ispersistMapIntResult = false;
```

[0099]
```
        protected boolean isStoreMapIntResult = false;
```

[0100]
```
        protected String jobId = null;
```

[0101]
```
        protected ClusterHosts clusterHosts = null;
```

[0102]
```
        protected HostSelector hostSelector = null;
```

[0103]
```
          public JobClient(NSName mapper, NSName reducer, NSName mapTaskName,
```

[0104]
```
int throttle,boolean isPersistMapIntResult)
```

[0105]
```
        {
```

[0106]
```
              this.mapper = mapper;
```

[0107]
```
              this.reducer = reducer;
```

[0108] this.mapTaskName = mapTaskName ;

[0109] this.throttle = throttle ;

[0110] this.isPersistMapIntResult = isPersistMapIntResult ;

[0111] jobConf = new JobConfiguration() ;

[0112] clusterHosts = new ClusterHosts() ;

[0113] hostSelector = new RoundRobinHostSelector() ;

[0114] hostSelector.setClusterHosts(clusterHosts) ;

[0115] jobConf.readJobConfiguration(clusterHosts) ;

[0116] jobId = UUID.generate() ;// 比整数好,因为在服务器重启前后它将是唯一的

[0117] }

[0118] public JobClient(NSName mapper, NSName reducer, NSName mapTaskName,

[0119] int throttle, boolean isPersistMapIntResult, boolean isStoreMapIntResult)

[0120] {

[0121] this(mapper, reducer, mapTaskName, throttle,

[0122] isPersistMapIntResult) ;

[0123] this.isStoreMapIntResult = isStoreMapIntResult ;

[0124] }

[0125] public void submitJob(IData pipeline)

[0126] {

[0127] jobInProgress = new JobInProgress(this, pipeline) ;

[0128] jobInProgress.executeAndTrackJob() ;

[0129] }

[0130] }

[0131] 可以看出,当 JobClient 的新的实例通过调用其构造器而被创建时(参见第 15 页第 17 行),它以参数 mapper(要用于当前作业的映射器实现)、reducer(要使用的化简器实现)、throttle(期望的并行服务执行的数目)以及 isPersistMapIntResult(中间结果是应当被存储在从属服务器的存储器上还是存储在持续性的文件系统中)作为输入。当调用 submitJob() 方法时(参见第 16 页第 4 行),该方法取得类型为 IData 的 pipeline 参数,该参数优选地包括要处理的输入数据,例如 EDI 文档的数据。submitJob() 随后创建新的 JobInProgress 实例并且调用其 executeAndTrackJob() 方法。

[0132] JobInProgress 的示例性实现在以下代码列表中示出 :

[0133] package com.wm.app.b2b.server.mapred ;

[0134] import java.util.ArrayList ;

[0135] import com.wm.app.b2b.server.InvokeState ;

[0136] import com.wm.app.b2b.server.Service ;

[0137] import com.wm.app.b2b.server.Session ;

```
[0138] import com.wm.app.b2b.server.ThreadManager;
[0139] import com.wm.data.IData;
[0140] import com.wm.data.IDataCursor;
[0141] import com.wm.data.IDataUtil;
[0142] public class JobInProgress implements Runnable
[0143] {
[0144]       private TaskInProgress taskInProgress = null;
[0145]     private int mapTaskID = 0;
[0146]     private int reduceTaskID = 0;
[0147]     protected JobClient jobClient = null;
[0148]     protected TaskTracker taskTracker = new TaskTracker();
[0149]     private InvokeState invokeState = null;
[0150]     private Session session = null;
[0151]     private boolean isJobDone = false;
[0152]     IData mapperPipeline = null;
[0153]     public JobInProgress(JobClient jobClient,IData mapperPipeline)
[0154]     {
[0155]           this.jobClient = jobClient;
[0156]           this.mapperPipeline = mapperpipeline;
[0157]     }
[0158]     public void executeAndTrackJob()
[0159]     {
[0160]           if(InvokeState.getCurrentState() != null){
[0161]                 invokeState =
[0162] (InvokeState)InvokeState.getCurrentState().clone();
[0163]           }
[0164]           if(InvokeState.getCurrentSession() != null){
[0165]                 session =
[0166] (Session)InvokeState.getCurrentSession().clone();
[0167]           }
[0168]           //long startTime = System.currentTimeMillis();
[0169]           ThreadManager.runTarget(this);
[0170]           synchronized(this){
[0171]                 while(isJobDone == false){
[0172]                       try{
[0173]                             this.wait();
[0174]                       }catch(InterruptedException e){
[0175]                        }
[0176]                  }
```

```
[0177]              }
[0178]              //long endTime = System.currentTimeMillis() ;
[0179]                   //System.out.println(  " The time taken in mill
secs:" +(endTime
[0180]   -startTime)) ;
[0181]       }
[0182]       public void run()
[0183]       {
[0184]              boolean isAllMapTasksDispatched = false ;
[0185]              ArrayList<Integer>mapTaskIdsCompleteList = null ;
[0186]              Integer[]mapTaskIdsComplete = null ;
[0187]              IData reducerPipeline = mapperPipeline ;
[0188]              IDataCursor reducerPipelineCur = null ;
[0189]              InvokeState.setCurrentState(invokeState) ;
[0190]              InvokeState.setCurrentSession(session) ;
[0191]              taskInProgress = new TaskInProgress() ;
[0192]              int numMapTasksComplete = 0 ;
[0193]              while(true)
[0194]              {
[0195]                    //检查是否分派了所有映射任务
[0196]                    if(！ isAllMapTasksDispatched){
[0197]                          //分割器
[0198]                          mapTaskID++ ;
[0199]                          IDataCursor mapperPipelineCursor =
[0200]   mapperPipeline.getCursor() ;
[0201]                          IDataUtil.put(mapperPipelineCursor,
[0202]   MapReduceConstants.MAP_ITERATION,mapTaskID) ;
[0203]                          try {
[0204]                                Service.doInvoke(jobClient.mapper,
[0205]   mapperPipeline) ;
[0206]                          }catch(Exception e){
[0207]                              e.printStackTrace() ;
[0208]                              break ;
[0209]                          }
[0210]                          //端点服务
[0211]                          IData serviceInputData =
[0212]   (IData)IDataUtil.get(mapperPipelineCursor,
[0213]   MapReduceConstants.SERVICE_INPUT_DATA) ;
[0214]                          IDataUtil.remove(mapperPipelineCursor,
```

```
[0215]  MapReduceConstants.SERVICE_INPUT_DATA) ;
[0216]                          isAllMapTasksDispatched =
[0217]  IDataUtil.get(mapperPipelineCursor, MapReduceConstants.IS_LAST_SPLIT)
==
[0218]  null ? false:true ;
[0219]                          if(isAllMapTasksDispatched){
[0220]                              IDataUtil.remove(mapperPipelineCursor,
[0221]  MapReduceConstants.IS_LAST_SPLIT) ;
[0222]                              IDataUtil.remove(mapperPipelineCursor,
[0223]  MapReduceConstants.MAP_ITERATION) ;
[0224]                          }
[0225]                          mapperPipelineCursor.destroy() ;
[0226]                                  //System.out.println(" spawning map
task:" +
[0227]  mapTaskID) ;
[0228]                                   MapTask mapTask = TaskFactory.
createMapTask(this,
[0229]  mapTaskID) ;
[0230]                          mapTask.setTaskInput(serviceInputData) ;
[0231]                                  mapTask.setHostSelector(jobClient.
hostSelector) ;
[0232]                          //扼制并行执行的 mapTask 的最大数目
[0233]                          //它考虑了先前提交和完成的映射任务
[0234]                          synchronized(taskTracker){
[0235]                              try{
[0236]                                  numMapTasksComplete =
[0237]  taskTracker.getNumCompletedMapTasks() ;
[0238]                                   while((mapTaskID-numMapTasksComple
te)
[0239]  > jobClient.throttle){
[0240]                                                      //System.out.println
(" waiting for
[0241]  some map tasks to complete,num mapTasks onGoing:" +(mapTaskID-
[0242]  numMapTasksComplete)) ;
[0243]                                                  taskTracker.wait() ;
[0244]                                                  numMapTasksComplete =
[0245]  taskTracker.getNumCompletedMapTasks() ;
[0246]                                   }
[0247]                              }catch(InterruptedException e){
```

```
[0248]                                    //待做事项
[0249]                                }
[0250]                            }
[0251]                                taskInProgress.executeTask(mapTask,
invokeState,
[0252]    session);
[0253]                        }else if(jobClient.reducer ! = null&&
[0254]    isAllMapTasksDispatched) {
[0255]                            //化简器
[0256]                            synchronized(taskTracker){
[0257]                                mapTaskIdsCompleteList =
[0258]    taskTracker.getCompletedMapTasks();
[0259]                                while(mapTaskIdsCompleteList==null ||
[0260]    mapTaskIdsCompleteList.size() == 0){
[0261]                                    try  {
[0262]                                        taskTracker.wait();
[0263]                                    }catch(InterruptedException e){
[0264]                                    }
[0265]                                    mapTaskIdsCompleteList =
[0266]    taskTracker.getCompletedMapTasks();
[0267]                                }
[0268]                            }
[0269]                            if(mapTaskIdsCompleteList ! = null&&
[0270]    mapTaskIdsCompleteList.size() > 0){
[0271]                                mapTaskIdsComplete =
[0272]    mapTaskIdsCompleteList.toArray(new Integer[0]);
[0273]                                    reduceTaskID+ = mapTaskIdsComplete.
length;
[0274]                                    //System.out.println (" processing
reduce
[0275]    tasks:" +mapTaskIdsComplete.length);
[0276]                                if(mapTaskID == reduceTaskID){
[0277]                                    reducerPipelineCur =
[0278]    reducerPipeline.getCursor();
[0279]                                    IDataUtil.put(reducerPipelineCur,
[0280]    MapReduceConstants.IS_LAST_REDUCE_STEP,true);
[0281]                                    reducerPipelineCur.destroy();
[0282]                                }
[0283]                                ReduceTask reduceTask =
```

```
[0284]  TaskFactory.createReduceTask(this,reduceTaskID);
[0285]                                                  reduceTask.
setTaskInput(reducerPipeline);
[0286]          reduceTask.setCompletedMapTasks(mapTaskIdsComplete);
[0287]                                  reduceTask.runTask();
[0288]                                    //如果由于某种原因,化简任务之一返回了
null,
[0289]                                   //此结果不应当造成以前收集的所有其他化
简任务输出无效
[0290]                                   //待做事项:当整个框架完成时,失败的(返
回了 null 的)
[0291]                                  //化简任务应当被重新执行
[0292]                                  IData tempReduceOutput =
[0293]  reduceTask.getTaskResult();
[0294]                                  if(tempReduceOutput ! =null){
[0295]                                                  reducerPipeline =
tempReduceOutput;
[0296]                                  }
[0297]                              }
[0298]                              //所有映射和化简任务完成
[0299]                              if(isAllMapTasksDispatched && mapTaskID ==
[0300]  reduceTaskID){
[0301]                                  IDataCursor reduceServicePipelineCur =
[0302]  reducerPipeline.getCursor();
[0303]                                  IData reduceIntermediateOutput =
[0304]  (IData)IDataUtil.get(reduceServicePipelineCur,
[0305]  MapReduceConstants.SERVICE_OUTPUT_DATA);
[0306]                                   IDataUtil.remove(reduceServicePipelineC
ur,
[0307]  MapReduceConstants.SERVICE_OUTPUT_DATA);
[0308]                                   IDataUtil.remove(reduceServicePipelineC
ur,
[0309]  MapReduceConstants.IS_LAST_REDUCE_STEP);
[0310]                                  reduceServicePipelineCur.destroy();
[0311]                                  //将最终结果合并成管道数据
[0312]                                                  IDataUtil.
merge(reduceIntermediateOutput,
[0313]  mapperPipeline);
[0314]                                  break;
```

```
[0315]                              }
[0316]                          }
[0317]                  }
[0318]                  final String tabSpace ="          " ;
[0319]                  //打印映射任务和化简任务的状态
[0320]                          System.out.println(  " Num  Map
Tasks:" +mapTaskID+tabSpace+
[0321]   " Num Reduce Tasks:" +reduceTaskID) ;
[0322]                  //作业完成即清理
[0323]                  InvokeState.setCurrentState(null) ;
[0324]                  InvokeState.setCurrentSession(null) ;
[0325]                  jobClient.clusterHosts.freeClusterHosts() ;
[0326]                  synchronized(this){
[0327]                        isJobDone = true ;
[0328]                        this.notifyAll() ;
[0329]                  }
[0330]           }
[0331]   }
```

[0332] 可以看出,在此示例中 JobInProgress 的 run() 方法包括用于处理输入文件的主代码,即分割(参见第 18 页第 32 行)、执行“并行化”的 flow 服务(参见第 19 页第 8 行)和化简(参见第 20 页第 17 行)的步骤。

[0333] 执行映射的 MapTask 的示例性实现在以下代码列表中示出:

```
[0334]   package com.wm.app.b2b.server.mapred ;
[0335]   import com.wm.app.b2b.server.InvokeState ;
[0336]   import com.wm.app.b2b.server.ThreadManager ;
[0337]   import com.wm.data.IData ;
[0338]   import com.wm.lang.ns.NSName ;
[0339]   public class MapTask extends AbstractTask implements Runnable
[0340]   {
[0341]         public MapTask(int id,TaskTracker taskTracker,NSName mapSvcName,
[0342]   TaskResultStoreConfig policy)
[0343]         {
[0344]               super(id,taskTracker,mapSvcName,policy) ;
[0345]         }
[0346]         public void runTask()
[0347]         {
[0348]               ThreadManager.runTarget(this) ;
[0349]         }
[0350]         public void run()
```

```
[0351]          {
[0352]                  InvokeState.setCurrentState(invokeState) ;
[0353]                  InvokeState.setCurrentSession(session) ;
[0354]                  taskStatus = new TaskStatus() ;
[0355]                  IData output = null ;
[0356]                  try {
[0357]                              output = RPC.remoteInvoke(hostSelector.
getHostInfoEntry(),
[0358]    serviceName,taskInput) ;
[0359]                      taskInput = null ;
[0360]                  }catch(Exception e){
[0361]                      output = null ;
[0362]                  }
[0363]                  storeTaskResult(output) ;
[0364]              synchronized(taskTracker){
[0365]                      taskTracker.addMapTaskToCompletedList(this) ;
[0366]                      setTaskStatus(TaskStatus.TASK_COMPLETE) ;
[0367]                      taskTracker.notifyAll() ;
[0368]              }
[0369]               InvokeState.setCurrentState(null) ;
[0370]               InvokeState.setCurrentSession(null) ;
[0371]               //System.out.println(" map task time is:" +toalSvcTime) ;
[0372]          }
[0373]    }
```

[0374] 可以看出，当 MapTask 被执行时，即，当其 run() 方法被调用时，MapTask 调用 RPC 类的 remoteInvoke() 方法（参见第 23 页第 32 行），该 remoteInvoke() 方法取得三个输入参数 :hostSelector.getHostInfoEntry()、serviceName 和 taskInput。taskInput 是从超类 AbstractTask 继承来的属性并且优选地包括要处理的输入，例如 EDI 文档的数据。

[0375] RPC 及其 remoteInvoke() 方法的示例性实现在以下代码列表中示出 :

```
[0376]    package com.wm.app.b2b.server.mapred ;
[0377]    import com.wm.app.b2b.client.Context ;
[0378]    import com.wm.app.b2b.client.ServiceException ;
[0379]    import com.wm.app.b2b.server.ACLManager ;
[0380]    import com.wm.app.b2b.server.BaseService ;
[0381]    import com.wm.app.b2b.server.Service ;
[0382]    import com.wm.app.b2b.server.invoke.InvokeManager ;
[0383]    import com.wm.app.b2b.server.ns.Namespace ;
[0384]    import com.wm.lang.ns.NSName ;
[0385]    import com.wm.data.IData ;
```

```
[0386] import com.wm.data.IDataCursor;
[0387] import com.wm.data.IDataUtil;
[0388] public class RPC
[0389] {
[0390]       private static final String DEFAULT_RPC_SERVER =" localhost" ;
[0391]     private static final int DEFAULT_RPC_PORT = 5555;
[0392]     private static final String DEFAULT_RPC_UID =" Administrator" ;
[0393]     private static final String DEFAULT_RPC_PASSWD =" manage" ;
[0394]     private static String rpcServer = null;
[0395]     private static int rpcPort = -1;
[0396]     private static String rpcUID = null;
[0397]     private static String rpcPasswd = null ;
[0398]     private static Objectlock = new Object();
[0399]     public static IData remoteInvoke(HostInfo hostInfoEntry,NSName
[0400] mapTaskName,IData input)throws Exception
[0401]     {
[0402]       if(hostInfoEntry == null){
[0403]             return null;
[0404]       }
[0405]       boolean isPrimary = hostInfoEntry.isPrimary();
[0406]       if(isPrimary){
[0407]             return Service.doInvoke(mapTaskName,input);
[0408]       }else {
[0409]             Context context = null;
[0410]             //在这里同步,否则可能为主机创建不止一个上下文
[0411]             synchronized  (lock){
[0412]                   if(! hostInfoEntry.isConnected){
[0413]                               String hostName = hostInfoEntry.
getHostName();
[0414]                         int port = hostInfoEntry.getPort();
[0415]                         context = new Context();
[0416]          context.connect(hostName+" :" +port,DEFAULT_RPC_UID,DEFAULT_
RPC_PASSWD);
[0417]                         hostInfoEntry.setContext(context);
[0418]                         hostInfoEntry.setConnected(true);
[0419]                   }else  {
[0420]                         context = hostInfoEntry.getContext();
[0421]                   }
[0422]             }
```

```
[0423]                if(context ! = null){
[0424]                   return context.invoke(mapTaskName,input) ;
[0425]               }
[0426]         }
[0427]               return null ;
[0428]        }
[0429]       }
```

[0430] ReduceTask 的示例性实现在以下代码列表中示出 :

```
[0431] package com.wm.app.b2b.server.mapred ;
[0432] import com.wm.app.b2b.server.Service ;
[0433] import com.wm.data.IData ;
[0434] import com.wm.data.IDataCursor ;
[0435] import com.wm.data.IDataUtil ;
[0436] import com.wm.lang.ns.NSName ;
[0437] public class ReduceTask extends AbstractTask
[0438] {
[0439]       private Integer[] mapTaskIDArray = null ;
[0440]       private boolean inMemory = false ;
[0441]       public ReduceTask(int id,TaskTracker taskTracker,NSName
[0442] reduceSvcName,TaskResultStoreConfig resultStoragePolicy)
[0443]       {
[0444]             super(id,taskTracker,reduceSvcName,resultStoragepolicy) ;
[0445]                   if(resultStoragePolicy instanceof
TaskResultMemStoreConfig){
[0446]                   inMemory = true ;
[0447]             }
[0448]       }
[0449]       public void setCompletedMapTasks(Integer[]mapTaskIDArray){
[0450]             this.mapTaskIDArray = mapTaskIDArray ;
[0451]       }
[0452]       public void runTask()
[0453]       {
[0454]             if(mapTaskIDArray == null || mapTaskIDArray.length == 0)
{
[0455]                   return ;
[0456]             }
[0457]             if(inMemory){
[0458]                   invokeReduceService(mapTaskIDArray) ;
[0459]             }
```

```
[0460]              else {
[0461]                    IDataCursor reduceServicePipelineCur =
[0462]   taskInput.getCursor() ;
[0463]                    boolean isLastReduceBatch =
[0464]   IDataUtil.getBoolean(reduceServicePipelineCur,
[0465]   MapReduceConstants.IS_LAST_REDUCE_STEP) ;
[0466]                    IDataUtil.remove(reduceServicePipelineCur,
[0467]   MapReduceConstants.IS_LAST_REDUCE_STEP) ;
[0468]                    reduceServicePipelineCur.destroy() ;
[0469]                    for(int i = 0 ;i < mapTaskIDArray.length ;i++){
[0470]                         if(i == mapTaskIDArray.length-1 &&
[0471]   isLastReduceBatch){
[0472]                              reduceServicePipelineCur =
[0473]   taskInput.getCursor() ;
[0474]                              IDataUtil.put(reduceServicepipelineCur,
[0475]   MapReduceConstants.IS_LAST_REDUCE_STEP,true) ;
[0476]                              reduceServicepipelineCur.destroy() ;
[0477]                         }
[0478]                         invokeReduceService(newInteger[]
[0479]   {mapTaskIDArray[i]}) ;
[0480]                    }
[0481]              }
[0482]              storeTaskResult(taskInput) ;
[0483]         }
[0484]         private void invokeReduceService(Integer[]mapTaskIDs){
[0485]              IData[]reduceInputDataArray = null ;
[0486]                   IDataCursor reduceServicePipelineCur = taskInput.
getCursor() ;
[0487]              IData reduceIntermediateOutput =
[0488]   (IData)IDataUtil.get(reduceServicePipelineCur,
[0489]   MapReduceConstants.SERVICE_OUTPUT_DATA) ;
[0490]              int length = 0 ;
[0491]              if(reduceIntermediateOutput ! = null){
[0492]                    length = mapTaskIDs.length+1 ;
[0493]              }else {
[0494]                    length = mapTaskIDs.length ;
[0495]              }
[0496]              int count = 0 ;
[0497]              reduceInputDataArray = new IData[length] ;
```

```
[0498]              if(reduceIntermediateOutput ! = null){
[0499]                                  reduceInputDataArray[count++] =
reduceIntermediateOutput ;
[0500]              }
[0501]              for(Integer mapTaskID:mapTaskIDs){
[0502]                    synchronized(taskTracker){
[0503]                          reduceInputDataArray[count++] =
[0504]   taskTracker.removeMapTask(mapTaskID).getTaskResult() ;
[0505]                    }
[0506]              }
[0507]              IDataUtil.put(reduceServicePipelineCur,
[0508]   MapReduceConstants.REDUCE_INPUT_DATA,reduceInputDataArray) ;
[0509]              if(reduceInputDataArray ! = null){
[0510]                    try{
[0511]                          Service.doInvoke(serviceName,taskInput) ;
[0512]                    }catch(Exception e){
[0513]                          e.printStackTrace() ;
[0514]                    }
[0515]              }
[0516]              reduceInputDataArray = null ;
[0517]              IDataUtil.remove(reduceServicepipelineCur,
[0518]   MapReduceConstants.REDUCE_INPUT_DATA) ;
[0519]              reduceServicePipelineCur.destroy() ;
[0520]        }
[0521]   }
```

[0522] MapTask 和 ReduceTask 都以抽象类 AbstractTask 作为超类，即它们继承其属性以及 set 和 get 方法，这在以下 AbstractTask 的示例性代码列表中示出：

```
[0523]   package com.wm.app.b2b.server.mapred ;
[0524]   import com.wm.app.b2b.server.InvokeState ;
[0525]   import com.wm.app.b2b.server.Session ;
[0526]   import com.wm.data.IData ;
[0527]   import com.wm.lang.ns.NSName ;
[0528]   public abstract class AbstractTask implements Task
[0529]   {
[0530]        private int taskID = -1 ;
[0531]        protected TaskStatus taskStatus = null ;
[0532]        protected TaskTracker taskTracker = null ;
[0533]        protectedInvokeState invokeState = null ;
[0534]        protected Session session = null ;
```

```
[0535]        protected IData taskInput = null;
[0536]        protected NSName serviceName = null;
[0537]        protected HostSelector hostSelector = null;
[0538]        private TaskResultStoreConfig taskResultStoreCfg = null;
[0539]        public AbstractTask(int id,TaskTracker taskTracker,NSName
[0540]  serviceName,TaskResultStoreConfigtaskResultStoreCfg){
[0541]              this.taskID = id;
[0542]              this.taskTracker = taskTracker;
[0543]              this.serviceName = serviceName;
[0544]              this.taskResultStoreCfg = taskResultStoreCfg;
[0545]        }
[0546]        public void setInvokeState(InvokeState invokeState)
[0547]        {
[0548]              this.invokeState = invokeState;
[0549]        }
[0550]        public void setInvokeSession(Session session)
[0551]       {
[0552]              this.session = session;
[0553]        }
[0554]        public void setTaskID(int taskID)
[0555]        {
[0556]              this.taskID = taskID;
[0557]        }
[0558]        public int getTaskID()
[0559]        {
[0560]              return taskID;
[0561]        }
[0562]        public void setTaskInput(IData taskInput){
[0563]              this.taskInput = taskInput;
[0564]        }
[0565]              public void setTaskStatus(int status)
[0566]        {
[0567]              taskStatus.setTaskStatus(status);
[0568]        }
[0569]        public void storeTaskResult(IData result){
[0570]              this.taskResultStoreCfg.storeResult(result);
[0571]        }
[0572]        public IData getTaskResult(){
[0573]              return this.taskResultStoreCfg.fetchResultAndDestroy();
```

```
[0574]        }
[0575]        public void setHostSelector(HostSelector hostSelector)
[0576]        {
[0577]              this.hostSelector = hostSelector;
[0578]        }
[0579]  }
```

[0580] 可以看出，AbstractTask 本身实现接口 Task，其示例性实现在以下代码列表中示出：

```
[0581]  package com.wm.app.b2b.server.mapred;
[0582]  import com.wm.app.b2b.server.InvokeState ;
[0583]  import com.wm.app.b2b.server.Session;
[0584]  import com.wm.data.IData;
[0585]  public interface Task
[0586]  {
[0587]        public void setInvokeState(InvokeState invokeState);
[0588]        public void setInvokeSession(Session session);
[0589]        public void setHostSelector(HostSelector hostSelector);
[0590]        public intgetTaskID();
[0591]        public void setTaskID(int taskID);
[0592]        public void setTaskStatus(int status);
[0593]        public void setTaskInput(IData taskInput);
[0594]        public void runTask();
[0595]        public void storeTaskResult(IData result);
[0596]        public IData getTaskResult();
[0597]  }
```

[0598] 在本发明的示例性实现中需要若干个其他基础类和接口，它们在以下代码列表中示出：

```
[0599]  package com.wm.app.b2b.server.mapred;
[0600]  import java.util.ArrayList;
[0601]  import java.util.HashMap;
[0602]  public class TaskTracker
[0603]  {
[0604]        private ArrayList<Integer>completedMapTasks = new
[0605]  ArrayList<Integer>();
[0606]        private ArrayList<Integer>completedReduceTasks = new
[0607]  ArrayList<Integer>();
[0608]           private HashMap<Integer, MapTask>mapResults = new
HashMap<Integer,
[0609]  MapTask>();
```

```
[0610]        private HashMap<Integer,ReduceTask>reduceResults = new
[0611]  HashMap<Integer,ReduceTask>() ;
[0612]        public ArrayList<Integer>getCompletedMapTasks()
[0613]        {
[0614]              ArrayList<Integer>ret = completedMapTasks ;
[0615]              //清除原始列表,以便我们只保留新信息
[0616]              completedMapTasks = new ArrayList<Integer>() ;
[0617]              return ret ;
[0618]        }
[0619]        public int getNumCompletedMapTasks()
[0620]        {
[0621]              return completedMapTasks.size() ;
[0622]        }
[0623]        public void addMapTaskToCompletedList(MapTask task)
[0624]        {
[0625]              completedMapTasks.add(task.getTaskID()) ;
[0626]              mapResults.put(task.getTaskID(),task) ;
[0627]        }
[0628]        public MapTask removeMapTask(Integer taskID)
[0629]        {
[0630]              return mapResults.remove(taskID) ;
[0631]        }
[0632]        public ArrayList<Integer>getCompletedReduceTasks()
[0633]        {
[0634]              ArrayList<Integer>ret = completedReduceTasks ;
[0635]              //清除原始列表,以便我们只保留新信息
[0636]              completedReduceTasks = new ArrayList<Integer>() ;
[0637]              return ret ;
[0638]        }
[0639]      public int getNumCompletedReduceTasks()
[0640]      {
[0641]            return completedReduceTasks.size() ;
[0642]      }
[0643]      public void addReduceTaskToCompletedList(ReduceTask task)
[0644]      {
[0645]            completedReduceTasks.add(task.getTaskID()) ;
[0646]            reduceResults.put(task.getTaskID(),task) ;
[0647]      }
[0648]      public ReduceTask removeReduceTask(Integer taskID)
```

```
[0649]        {
[0650]              return reduceResults.remove(taskID) ;
[0651]        }
[0652]    }
[0653]    package com.wm.app.b2b.server.mapred ;
[0654]    / * *
[0655]     * TaskFactory 利用以下配置创建 MapTask 或 ReduceTask
[0656]     * 1. 若标志 isPersistMapIntResult 和 isStoreMapIntResult 被设置,中间映射输出将被
[0657]     *持续保留在 $IS-DIR/logs 目录中。文件名为 Map_jobId_taskId 或 Reduce_jobId_taskId
[0658]     * 2. 如果标志 isPersistMapIntResult 未被设置而标志 isStoreMapIntResult 被设置,则
[0659]     *中间映射输出将被保存在存储器中。
[0660]     * 3. 对于所有其他组合,中间映射输出不被存储在文件 / 存储器中
[0661]     * /
[0662]    public class TaskFactory{
[0663]           public static MapTask createMapTask(JobInProgress job, int taskId){
[0664]               String tag =" Map_" +job.jobClient.jobId+" _" +taskId ;
[0665]               if(job.jobClient.isPersistMapIntResult){
[0666]                     if(job.jobClient.isStoreMapIntResult){
[0667]                           return new MapTask(taskId, job.taskTracker,
[0668]    job.jobClient.mapTaskName, new TaskResultFileStoreConfig(tag)) ;
[0669]                     }
[0670]                     else {
[0671]                           return new MapTask(taskId, job.taskTracker,
[0672]    job.jobClient.mapTaskName, new TaskResultNoStoreConfig(tag)) ;
[0673]                     }
[0674]               }
[0675]               else{
[0676]                     if(job.jobClient.isStoreMapIntResult){
[0677]                           return new MapTask(taskId, job.taskTracker,
[0678]    job.jobClient.mapTaskName, new TaskResultMemStoreConfig(tag)) ;
[0679]                     }
[0680]                     else{
[0681]                         return new MapTask(taskId, job.taskTracker,
[0682]    job.jobClient.mapTaskName, new TaskResultNoStoreConfig(tag)) ;
[0683]                     }
```

```
[0684]              }
[0685]         }
[0686]         public static ReduceTask createReduceTask(JobInProgress job,int
[0687]   taskId){
[0688]                     String tag = " Reduce_ " +job.jobClient.
jobId+" _" +taskId;
[0689]              return new ReduceTask(taskId,job.taskTracker,
[0690]   job.jobClient.reducer,new TaskResultMemStoreConfig(tag));
[0691]         }
[0692]   }
[0693]   package com.wm.app.b2b.server.mapred;
[0694]   import com.wm.app.b2b.server.InvokeState;
[0695]   import com.wm.app.b2b.server.Session;
[0696]   public class TaskInProgress
[0697]   {
[0698]           public void executeTask(Task task,InvokeState invokeState,
Session
[0699]   session)
[0700]         {
[0701]              task.setInvokeState(invokeState);
[0702]              task.setInvokeSession(session);
[0703]              task.runTask();
[0704]         }
[0705]   }
[0706]   package com.wm.app.b2b.server.mapred;
[0707]   import java.io.File;
[0708]   import java.io.FileInputStream;
[0709]   import java.io.InputStream;
[0710]   import com.wm.app.b2b.server.Server;
[0711]   import com.wm.data.IData;
[0712]   import com.wm.util.coder.IDataXMLCoder;
[0713]   import com.wm.util.coder.XMLCoderWrapper;
[0714]   public class TaskResultFileStoreConfig implements
TaskResultStoreConfig{
[0715]         private File savedFile = null;
[0716]         private String dataTag = null;
[0717]         public TaskResultFileStoreConfig(String dataTag){
[0718]               this.dataTag = dataTag;
[0719]         }
```

```
[0720]          public IData fetchResultAndDestroy(){
[0721]            InputStream is = null;
[0722]            try {
[0723]                XMLCoderWrapper xc = new XMLCoderWrapper();
[0724]                is = new FileInputStream(savedFile);
[0725]                return xc.decode(is);
[0726]            } catch (Exception e){
[0727]                e.printStackTrace();
[0728]            } finally {
[0729]                try{
[0730]                      if(is ! = null){
[0731]                            is.close();
[0732]                      }
[0733]                      } catch(Throwable t){
[0734]                            t.printStackTrace();
[0735]                      } finally {
[0736]                             savedFile.delete();
[0737]                      }
[0738]            }
[0739]            return null;
[0740]          }
[0741]          public void storeResult(IData result){
[0742]            try {
[0743]                    savedFile = new File(Server.getLogDir(),dataTag+" .
xml" );
[0744]                IDataxMLCoder xc = newIDataXMLCoder();
[0745]                xc.writeToFile(savedFile,result);
[0746]            } catch(Exception e){
[0747]                e.printStackTrace();
[0748]            }
[0749]          }
[0750]   }
[0751]   package com.wm.app.b2b.server.mapred;
[0752]   import com.wm.data.IData;
[0753]   public class TaskResultMemStoreConfig implements
TaskResultStoreConfig{
[0754]          private IData result = null;
[0755]          private String dataTag = null;
[0756]          public TaskResultMemStoreConfig(String dataTag){
```

```
[0757]              this.dataTag = dataTag;
[0758]        }
[0759]        public IData fetchResultAndDestroy(){
[0760]              return result;
[0761]        }
[0762]        public void storeResult(IData result){
[0763]              this.result = result;
[0764]        }
[0765]  }
[0766]  package com.wm.app.b2b.server.mapred;
[0767]  import com.wm.data.IData;
[0768]  import com.wm.data.IDataFactory;
[0769]  public class TaskResultNoStoreConfig implements TaskResultStoreConfig{
[0770]        private String dataTag = null;
[0771]        public TaskResultNoStoreConfig(String dataTag){
[0772]              this.dataTag = dataTag;
[0773]        }
[0774]        public IData fetchResultAndDestroy(){
[0775]              returnIDataFactory.create();
[0776]        }
[0777]        public void storeResult(IData result){
[0778]        }
[0779]  }
[0780]  package com.wm.app.b2b.server.mapred;
[0781]  import com.wm.data.IData;
[0782]  public interface TaskResultStoreConfig{
[0783]        public void storeResult(IData result);
[0784]        public IData fetchReSultAndDestroy();
[0785]  }
[0786]  package com.wm.app.b2b.server.mapred;
[0787]  public class TaskStatus
[0788]  {
[0789]         public static final int TASK_DISPATCHED = 1;
[0790]        public static final int TASK_RUNNING = 2;
[0791]        public static final int TASK_COMPLETE = 3;
[0792]        private int taskStatus = 0;
[0793]        public int getTaskStatus()
[0794]        {
[0795]              return taskStatus;
```

```
[0796]          }
[0797]          public void setTaskStatus(int taskStatus)
[0798]          {
[0799]                this.taskStatus = taskStatus;
[0800]          }
[0801]    }
[0802]    package com.wm.app.b2b.server.mapred;
[0803]    public class RoundRobinHostSelector implements HostSelector
[0804]    {
[0805]          private ClusterHosts clusterHosts = null;
[0806]          private int hostIndex = 0;
[0807]        private Object lock = new Object();
[0808]          public void setClusterHosts(ClusterHosts clusterHosts)
[0809]          {
[0810]                this.clusterHosts = clusterHosts;
[0811]          }
[0812]          public synchronized HostInfo getHostInfoEntry()
[0813]          {
[0814]                HostInfo hostInfoEntry = null;
[0815]          synchronized(lock){
[0816]                if(clusterHosts.hostInfoArrayList.size() == 0){
[0817]                      return null;
[0818]                }
[0819]                     hostInfoEntry = clusterHosts.hostInfoArrayList.
get(hostIndex);
[0820]                hostIndex++;
[0821]               if(hostIndex == clusterHosts.hostInfoArrayList.size()){
[0822]                      hostIndex = 0;
[0823]               }
[0824]               }
[0825]          return hostInfoEntry;
[0826]          }
[0827]    }
[0828]    package com.wm.app.b2b.server.mapred;
[0829]    public class MapReduceConstants
[0830]    {
[0831]             public static final String SERVICE_INPUT_DATA
=" serviceInputData";
[0832]          public static final String IS_LAST_SPLIT  =" isLastSplit";
```

```
[0833]        public static final String MAP_ITERATION =" $mapIteration" ;
[0834]           public static final String IS_LAST_REDUCE_STEP
=" isLastReduceStep" ;
[0835]           public static final String REDUCE_INPUT_DATA
=" reduceInputData" ;
[0836]           public static final String SERVICE_OUTPUT_DATA
=" serviceOutputData" ;
[0837]        public static final String SERVER_ID =" id" ;
[0838]        public static final String SERVER_HOST =" host" ;
[0839]        public static final String SERVER_PORT =" port" ;
[0840]        public static final String IS_PRIMARY =" isPrimary" ;
[0841]        public static final String MAPPER =" mapper" ;
[0842]        public static final String REDUCER =" reducer" ;
[0843]        public static final String MAPTASKNAME =" mapTaskName" ;
[0844]        public static final String THROTTLE =" throttle" ;
[0845]  }
[0846]  package com.wm.app.b2b.server.mapred ;
[0847]  import java.io.File ;
[0848]  import java.io.Fi leInputStream ;
[0849]  import java.io.InputStream ;
[0850]  import java.util.Enumeration ;
[0851]  import java.util.Properties ;
[0852]      import java.util.StringTokenizer ;
[0853]      import com.wm.app.b2b.server.Server ;
[0854]      public class JobConfiguration
[0855]      {
[0856]           private static final String CLUSTERED_HOSTS_FILE_NAME =
[0857]  " ClusterHosts.cnf" ;
[0858]           public void readJobConfiguration(ClusterHosts clusterHosts)
[0859]           {
[0860]                try{
[0861]                     //不改变的默认值。
[0862]                     InputStream fis =new FileInputStream(new File
[0863]  (Server.getConfDir(),CLUSTERED_HOSTS_FILE_NAME)) ;
[0864]                     Properties prop =new Properties() ;
[0865]                     prop.load(fis) ;
[0866]                          Enumeration propertyNames = prop.
propertyNames() ;
[0867]                     while(propertyNames.hasMoreElements()){
```

```
[0868]                                  String key = (String)propertyNames.
nextElement() ;
[0869]                                  StringTokenizer values = new
[0870]   StringTokenizer(prop.getProperty(key), " ;:" ) ;
[0871]                                  clusterHosts.addNodeToCluster(key,
[0872]   values.nextToken(),Integer.parseInt(values.nextToken()),
[0873]   Boolean.parseBoolean(values.nextToken())) ;
[0874]                          }
[0875]              }catch(Throwable t){
[0876]                          //待做事项
[0877]                          t.printStackTrace() ;
[0878]                             System.out.println(" Error while reading the
clustered
[0879]   hosts properties file" ) ;
[0880]              }
[0881]          }
[0882]   }
[0883]   package com.wm.app.b2b.server.mapred ;
[0884]   public interface HostSelector
[0885]   {
[0886]          public void setClusterHosts(ClusterHosts clusterHosts) ;
[0887]          public HostInfo getHostInfoEntry() ;
[0888]   }
[0889]   package com.wm.app.b2b.server.mapred ;
[0890]   import com.wm.app.b2b.client.Context ;
[0891]   public class HostInfo
[0892]   {
[0893]          boolean isPrimary = false ;
[0894]          String hostDnsName = null ;
[0895]          int port = -1 ;
[0896]          boolean isConnected = false ;
[0897]          Context context = null ;
[0898]          public HostInfo(String hostDnsName,int port,boolean isPrimary)
[0899]          {
[0900]                 this.hostDnsName = hostDnsName ;
[0901]                 this.port = port ;
[0902]                 this.isPrimary = isPrimary ;
[0903]          }
[0904]          public String getHostName()
```

```
[0905]        {
[0906]              return hostDnsName ;
[0907]        }
[0908]        public int getPort()
[0909]        {
[0910]              return port ;
[0911]        }
[0912]        public boolean isPrimary()
[0913]        {
[0914]              return isPrimary ;
[0915]        }
[0916]        public boolean isConnected()
[0917]        {
[0918]              return isConnected ;
[0919]        }
[0920]        public void setConnected(boolean isConnected)
[0921]        {
[0922]              this.isConnected = isConnected ;
[0923]        }
[0924]        public Context getContext()
[0925]        {
[0926]              return context ;
[0927]        }
[0928]        public void setContext(Context  context)
[0929]        {
[0930]              this.context = context ;
[0931]        }
[0932]  }
[0933]  package com.wm.app.b2b.server.mapred ;
[0934]  import java.util.ArrayList ;
[0935]  import java.util.Iterator ;
[0936]  import java.util.Map ;
[0937]  import java.util.LinkedHashMap ;
[0938]  import com.wm.app.b2b.client.Context ;
[0939]  public class ClusterHosts
[0940]  {
[0941]          protected Map<String, HostInfo>clusterMap = new
LinkedHashMap<String,
[0942]  HostInfo>() ;
```

```
[0943]        protected ArrayList<HostInfo>hostInfoArrayList = new
[0944]   ArrayList<HostInfo>() ;
[0945]        public void addNodeToCluster(String nodeName, String hostDnsName,
int
[0946]   port, boolean isPrimary)
[0947]        {
[0948]           HostInfo hostInfo = new HostInfo(hostDnsName, port, isPrimary) ;
[0949]           clusterMap.put(nodeName, hostInfo) ;
[0950]           hostInfoArrayList.add(hostInfo) ;
[0951]     }
[0952]     public void freeClusterHosts()
[0953]     {
[0954]           clusterMap.clear() ;
[0955]           Iterator iter = hostInfoArrayList.iterator() ;
[0956]           Context context = null ;
[0957]           while(iter.hasNext()){
[0958]                 HostInfo hostInfo = (HostInfo)iter.next() ;
[0959]                 if(hostInfo ! = null){
[0960]                       if( ! hostInfo.isPrimary){
[0961]                             context = hostInfo.getContext() ;
[0962]                             context.disconnect() ;
[0963]                             hostInfo.setContext(null) ;
[0964]                             hostInfo.setConnected(false) ;
[0965]                       }
[0966]                 }
[0967]           }
[0968]           hostInfoArrayList.clear() ;
[0969]     }
[0970]   }
```

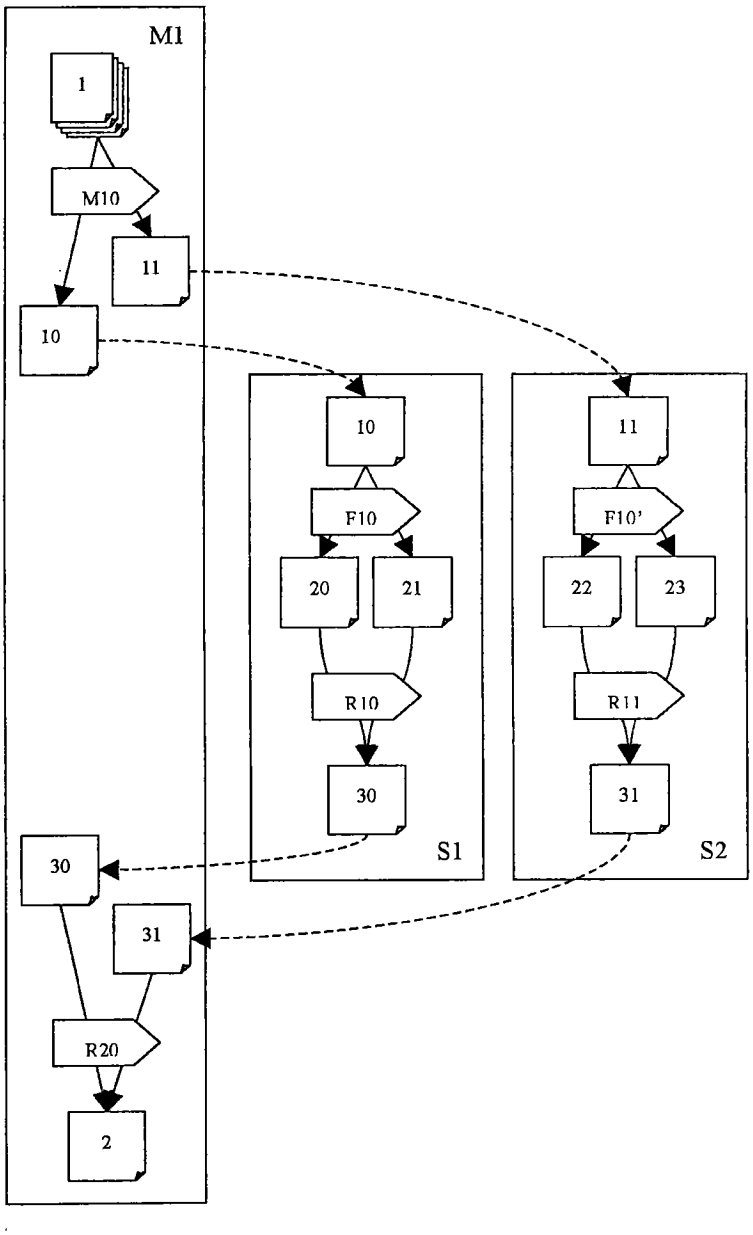

M1
1
M10
11
10
10
F10
20
21
R10
30
S1
11
F10'
22
23
R11
31
S2
30
31
R20
2

图 1

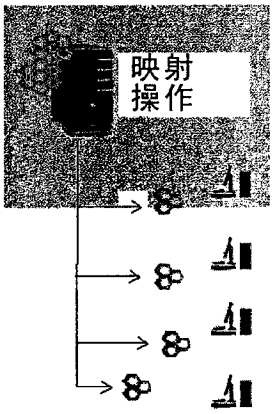

映射
操作

图 2

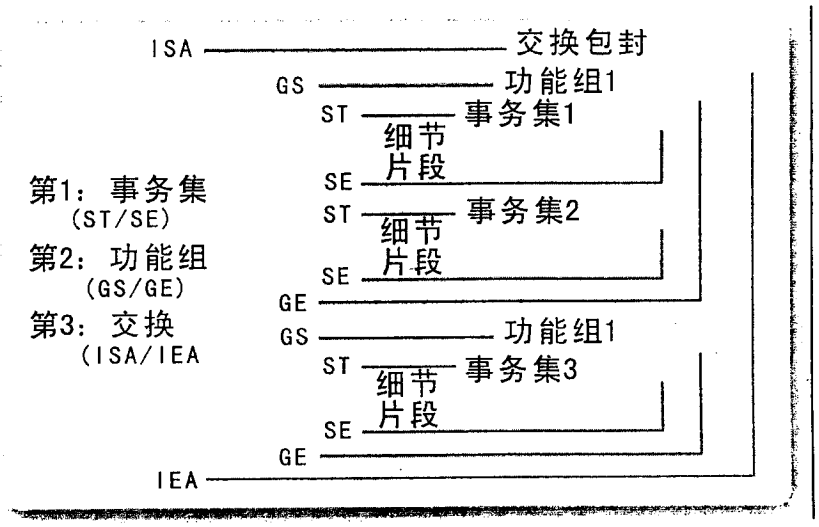

ISA
交换包封
GS
功能组1
ST
事务集1
细节片段
SE
ST
事务集2
细节片段
SE
GE
GS
功能组1
ST
事务集3
细节片段
SE
GE
IEA
第1：事务集 (ST/SE)
第2：功能组 (GS/GE)
第3：交换 (ISA/IEA

图 3

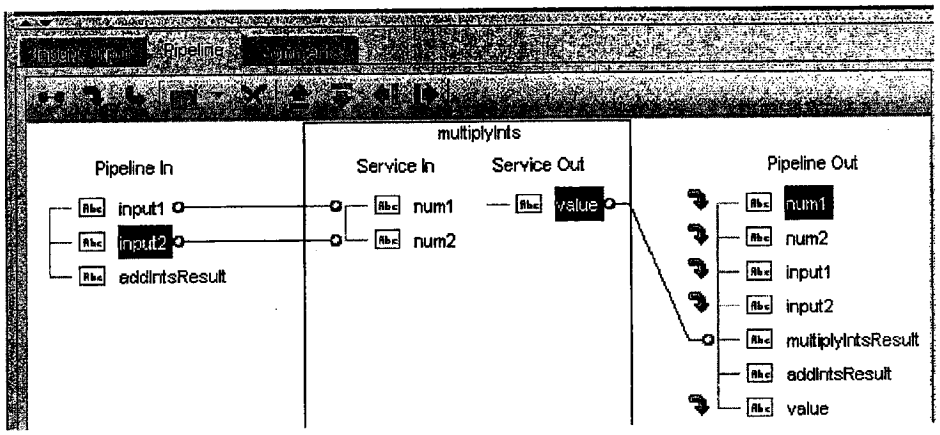

Pipeline
multiplyInts
Pipeline In
Service In
Service Out
Pipeline Out
input1
input2
addIntsResult
num1
num2
value
num1
num2
input1
input2
multiplyIntsResult
addIntsResult
value

图 4

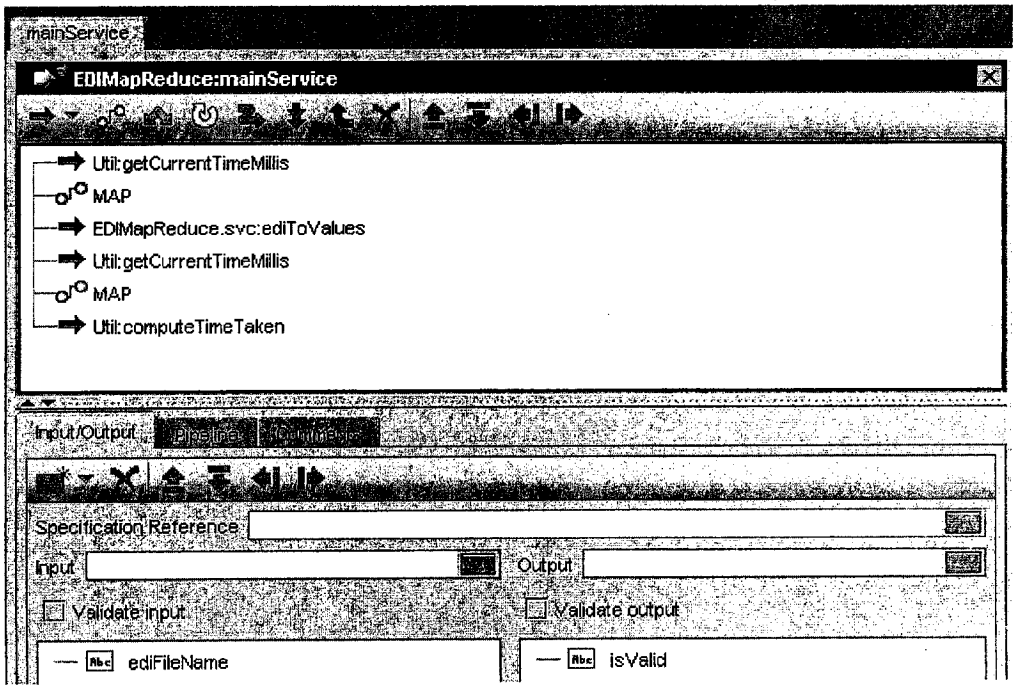

mainService
EDIMapReduce:mainService
Util:getCurrentTimeMillis
MAP
EDIMapReduce.svc:ediToValues
Util:getCurrentTimeMillis
MAP
Util:computeTimeTaken
Input/Output
Specification Reference
Input
Output
Validate input
Validate output
ediFileName
isValid

图 5

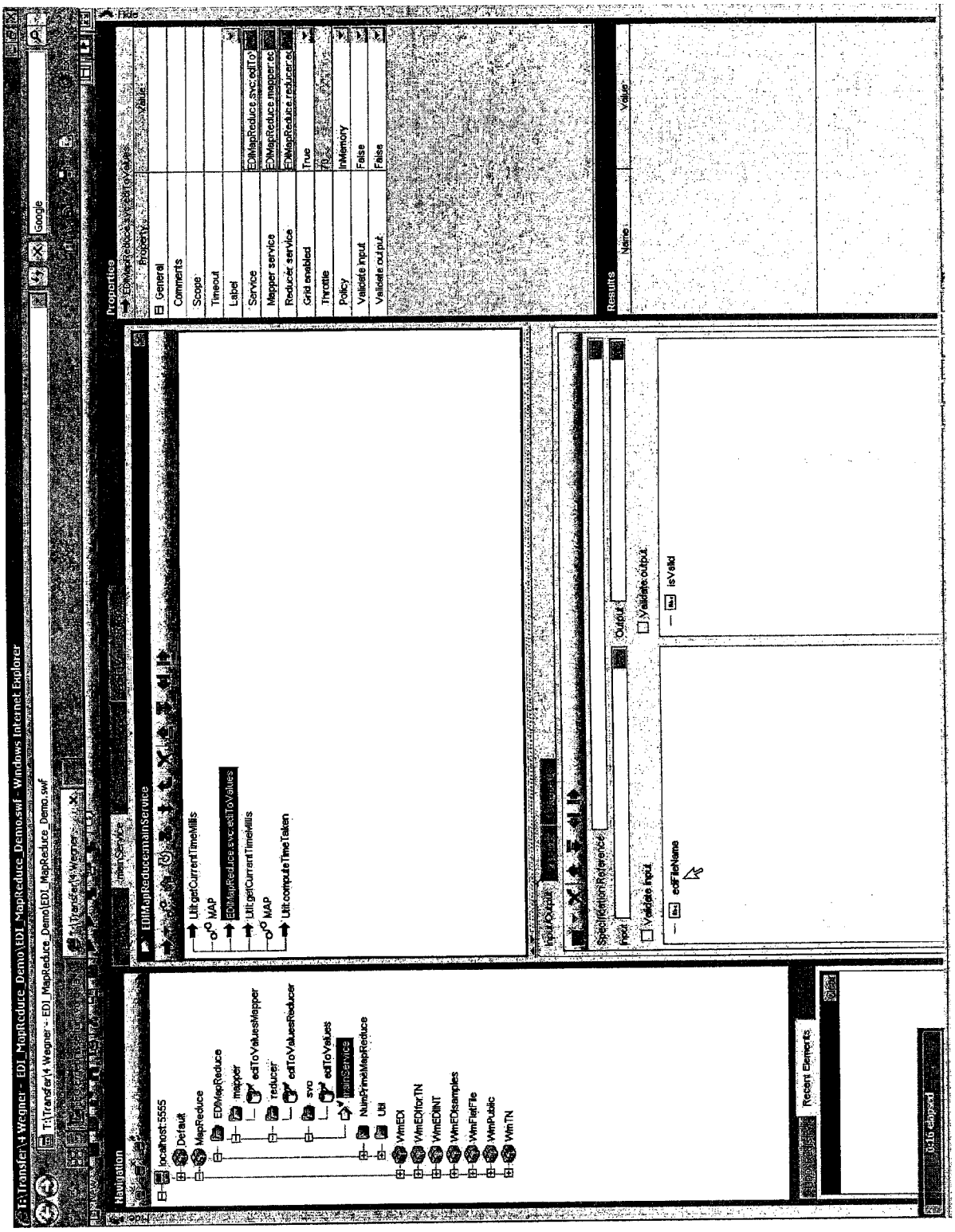

T:\Transfer\4 Wegner - EDI_MapReduce_Demo\EDI_MapReduce_Demo.swf - Windows Internet Explorer
Navigation
localhost:5555
Default
MapReduce
EDIMapReduce
mapper
edToValuesMapper
reducer
edToValuesReducer
svc
edToValues
mainService
NumPrimeMapReduce
Util
WmEDI
WmEDIforTN
WmEDINT
WmEDIsamples
WmFlatFile
WmPublic
WmTN
EDIMapReduce:mainService
Util:getCurrentTimeMillis
MAP
EDIMapReduce.svc:edIToValues
Util:getCurrentTimeMillis
MAP
Util:computeTimeTaken
Properties
Results
Name
Value
Validate input
edFileName
Validate output
isValid
Recent Elements
0.16 elapsed

图 6

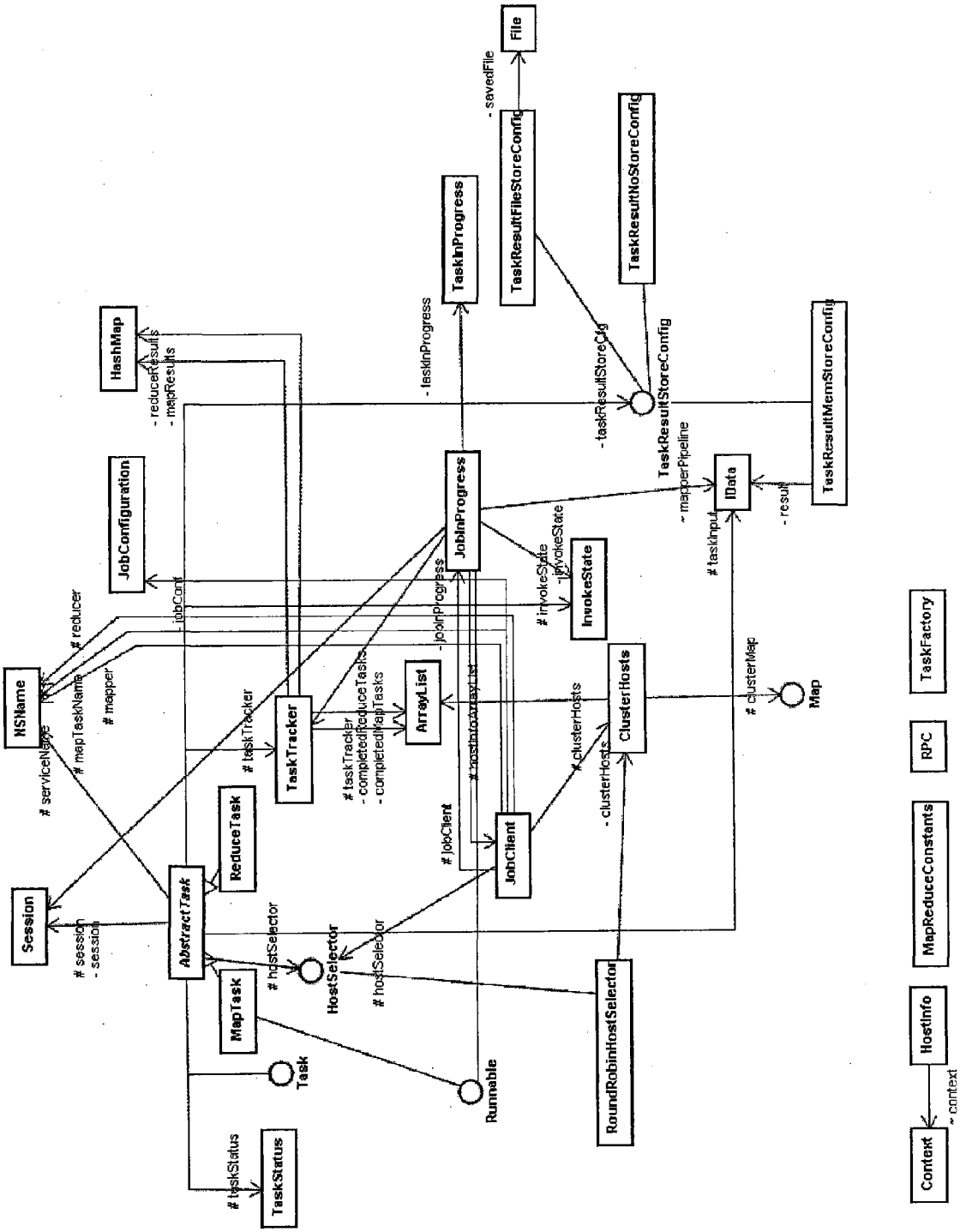

File
- savedFile
TaskResultFileStoreConfig
TaskResultNoStoreConfig
TaskInProgress
- taskInProgress
HashMap
- reduceResults
- mapResults
- taskResultStoreCfg
TaskResultStoreConfig
TaskResultMemStoreConfig
JobConfiguration
JobInProgress
~ mapperPipeline
IData
taskInput
- result
invokeState
- invokeState
InvokeState
- jobInProgress
reducer
mapper
mapTaskName
serviceName
NSName
taskTracker
TaskTracker
taskTracker
- completedReduceTasks
- completedMapTasks
ArrayList
clusterHosts
ClusterHosts
clusterMap
Map
TaskFactory
RPC
MapReduceConstants
HostInfo
~ context
Context
JobClient
JobClient
- clusterHosts
ReduceTask
Session
session
- session
AbstractTask
hostSelector
HostSelector
hostSelector
MapTask
Task
Runnable
RoundRobinHostSelector
taskStatus
TaskStatus

图 7