(19)中华人民共和国国家知识产权局

(12)发明专利

(10)授权公告号 CN 103403700 B
(45)授权公告日 2016.02.03

(21)申请号 201280010904.1
(22)申请日 2012.02.29
(30)优先权数据
13/038,086 2011.03.01 US
(85)PCT国际申请进入国家阶段日
2013.08.29
(86)PCT国际申请的申请数据
PCT/CA2012/050123 2012.02.29
(87)PCT国际申请的公布数据
WO2012/116449 EN 2012.09.07
(73)专利权人 国际商业机器公司
地址 美国纽约
(72)发明人 S·沃伊特亚娜桑 田媛媛 A·戈汀
D·R·博迪克 E·P·佩德纳特
B·赖因瓦尔德 V·辛杜瓦纳
S·塔提孔达 R·克里希纳穆尔塞
(74)专利代理机构 北京市中咨律师事务所
11247
代理人 于静 张亚非
(51)Int.Cl.
G06F 15/18(2006.01)
G06F 15/16(2006.01)
G06F 17/30(2006.01)
(56)对比文件
US 2011/0047172 A1,2011.02.24, 全文.
审查员 李江

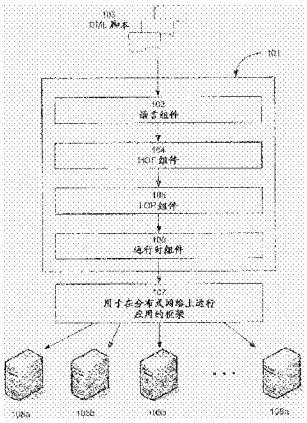
权利要求书2页 说明书10页 附图10页

(54)发明名称

用于在MAPREDUCE环境中处理机器学习算法的系统和方法

(57)摘要

描述了用于在MapReduce环境中处理机器学习(ML)算法的系统和方法。在方法的一个实施例中,所述方法包括接收要在所述MapReduce环境中执行的ML算法。所述方法还包括将所述ML算法解析为顺序的多个语句块,其中每个语句块包括多个基本运算(hop)。所述方法还包括自动确定每个语句块的执行计划,其中所述执行计划中的至少一个执行计划包括一个或多个低级运算(lop)。所述方法还包括以所述多个语句块的所述顺序实现所述执行计划。



CN 103403700 B

1. 一种用于在MapReduce环境中处理机器学习ML算法的计算机实现的方法，所述方法包括：

接收要在所述MapReduce环境中执行的ML算法；

将所述ML算法解析为顺序的多个语句块，其中每个语句块包括多个基本运算hop；

自动确定每个语句块的执行计划，其中所述执行计划中的至少一个执行计划包括一个或多个低级运算lop，其中确定每个语句块的执行计划包括将hop转变成至少一个lop；以及

以所述多个语句块的所述顺序实现所述执行计划。

2. 根据权利要求1的计算机实现的方法，其中确定所述语句块中的至少一个语句块的执行计划包括：

分析所述语句块以便确定所述多个hop之间的互连；以及

使用所述互连，通过矩阵和标量中的至少一个创建所述多个hop的有向非循环图HOPDag。

3. 根据权利要求2的计算机实现的方法，其中确定每个语句块的执行计划进一步包括：将所述HOPDag转变成包括多个lop的至少一个低级物理执行计划LOPDag。

4. 根据权利要求3的计算机实现的方法，其中所述LOPDag进一步包括标量运算的表示。

5. 根据权利要求1的计算机实现的方法，其中将所述hop转变成至少一个lop包括：

确定与所述hop等价的多组lop；以及

使用基于成本的优化从所述多组lop选择关联成本低于所述多组lop中的至少一个的关联成本的一组lop。

6. 根据权利要求3的计算机实现的方法，其中确定每个语句块的执行计划进一步包括：将所述LOPDag转换成MapReduce作业的代表性工作流，其中转换所述LOPDag包括判定是否可以由一个MapReduce作业来表示多个lop。

7. 根据权利要求6的计算机实现的方法，其中确定每个语句块的执行计划进一步包括优化所述HOPDag，其中所述HOPDag的优化包括代数重写和选择中间矩阵的物理表示中的至少一个。

8. 根据权利要求6的计算机实现的方法，其中确定每个语句块的执行计划进一步包括：通过使用基于成本的优化，在将HOPDag转变成LOPDag期间优化所述LOPDag。

9. 根据权利要求1的计算机实现的方法，其中实现所述执行计划包括：通过控制程序和集群上的MapReduce作业的多个工作流来管理标量运算的执行。

10. 根据权利要求9的计算机实现的方法，其中所述MapReduce作业的多个工作流的执行包括：使用从中创建所述工作流的语句块类型来判定是应在执行中跳过所述工作流还是应在执行中重复所述工作流。

11. 根据权利要求1的计算机实现的方法，还包括：判定是否可以由一个MapReduce作业来表示多个lop，其中此类判定包括基于lop的特征而在一个MapReduce作业中捎带所述lop。

12. 根据权利要求11的计算机实现的方法，还包括：创建所述MapReduce作业，其中从划分成至少三个阶段的通用MapReduce作业来实例化所述MapReduce作业，根据捎带方法

来参数化每个阶段以执行多个 lop。

13. 根据权利要求 1 的计算机实现的方法，还包括：优化所述执行计划，包括优化用于实现所述 ML 算法的数据块表示的大小。

14. 根据权利要求 13 的计算机实现的方法，其中优化所述数据块表示的大小依赖于每个数据块的局部稀疏性。

15. 一种用于在 MapReduce 环境中处理机器学习 ML 算法的系统，包括：

用于接收要在所述 MapReduce 环境中执行的 ML 算法的装置；

用于将所述 ML 算法解析为顺序的多个语句块的装置，其中每个语句块包括多个基本运算 hop；

用于自动确定每个语句块的执行计划的装置，其中所述执行计划中的至少一个执行计划包括一个或多个低级运算 lop，其中确定每个语句块的执行计划包括将 hop 转变成至少一个 lop；以及

用于以所述多个语句块的所述顺序实现所述执行计划的装置。

16. 根据权利要求 15 的系统，其中：

用于确定每个语句块的执行计划的装置包括用于通过矩阵、向量和标量中的至少一个创建所述多个 hop 的有向非循环图 HOPDag 的装置；

用于确定每个语句块的执行计划的装置包括用于将所述 HOPDag 转变成包括多个 lop 的至少一个低级物理执行计划 LOPDag 的装置；以及

用于将所述 HOPDag 转变成至少一个 LOPDag 的装置包括用于针对在所述 HOPDag 中实现的每个 hop 而将该 hop 转变成至少一个 lop 的装置；以及用于确定每个语句块的执行计划的装置包括用于将所述 lop 转换成 MapReduce 作业的多个工作流的装置，其中转换所述 lop 包括在一个 MapReduce 作业中捎带多个 lop。

用于在 MAPREDUCE 环境中处理机器学习算法的系统和方法

技术领域

[0001] 本公开的实施例一般地涉及数据处理系统领域。例如，本公开的实施例涉及用于在 MapReduce 环境(例如，Apache® Hadoop!)中处理机器学习算法的系统和方法。

背景技术

[0002] 越来越多地针对数据集使用机器学习(ML)算法以便提取和分析信息。对于诸如主题建模、推荐系统和因特网搜索查询之类的应用而言，随着数据集大小的增加，需要针对大型数据集可扩展地实现 ML 算法。目前的 ML 算法实现需要手动调整专用硬件，并且必须手动实现用于针对机器集群并行执行个体学习算法的方法。

[0003] 并行处理用于提高执行速度并增加要处理的数据量。但是，使用分布式网络或多个处理器意味着针对某个作业将存在更多的可能执行策略。一个问题是从多个策略(特别是用于实现多个 ML 算法的策略)选择良好执行策略的任务落在了程序员肩上。

发明内容

[0004] 描述了用于在 MapReduce 环境中处理机器学习(ML)算法的系统和方法。在一个实施例中，所述方法包括接收要在所述 MapReduce 环境中执行的 ML 算法。所述方法还包括将所述 ML 算法解析为顺序的多个语句块，其中每个语句块包括多个基本运算(hop)。所述方法还包括自动确定每个语句块的执行计划，其中所述执行计划中的至少一个执行计划包括一个或多个低级运算(lop)。所述方法还包括以所述多个语句块的所述顺序实现所述执行计划。

[0005] 提及该示例性实施例并不是为了限制或限定本发明，而是为了提供实例以便帮助理解本发明。将在“具体实施方式”中讨论示例性实施例，并在其中提供本公开的进一步描述。可以通过查看本说明书进一步理解本公开的不同实施例所提供的优点。

附图说明

[0006] 当参考附图阅读以下“具体实施方式”时，将更好地理解本发明的这些和其它特性、方面和优点，这些附图是：

[0007] 图 1 示出用于在 MapReduce 环境中实现机器学习算法的示例性系统；

[0008] 图 2 示出由图 1 的系统在 MapReduce 环境中实现机器学习算法时执行的示例性方法；

[0009] 图 3 示出图 1 中所示的执行图 2 中所示方法的系统的简单语句 A=B*(C/D) 的实例处理；

[0010] 图 4 示出如图 2 中所示方法的用于生成解析表示的示例性方法；

[0011] 图 5 示出由图 1 中所示的执行图 2 中所示方法的系统创建的基本运算的实例有向非循环图；

[0012] 图 6 示出表示 C/D 的二元 hop 的对应 lop 的一个实例；

[0013] 图 7 示出由图 1 中所示的执行图 2 中所示方法的系统将 lop 打包成 MapReduce 作业的一个实例；

[0014] 图 8 示出基于复制的矩阵乘法算法；

[0015] 图 9 示出基于叉积的矩阵乘法算法；

[0016] 图 10 示出用于实现图 1 中所示的系统和图 2 中所示的方法的实例计算机架构。

具体实施方式

[0017] 本公开的实施例一般地涉及数据处理系统领域。例如，本公开的实施例涉及用于在 MapReduce 环境中处理机器学习算法的系统和方法。在整个说明书中，出于解释目的，提供了大量特定的细节以便彻底理解本公开。但是，对本领域的技术人员显而易见的是，可以在没有其中某些特定细节的情况下实现本公开。在其它情况下，在框图中示出公知的结构和设备以避免使本公开的基本原理变得模糊不清。

[0018] MapReduce 是用于大型机器集群的通用并行编程范例。结合日益增长的针对大规模数据集运行机器学习(ML)算法的需要，本公开描述了用于针对 MapReduce 实现 ML 算法的新颖方法和系统。本公开描述了多种方法和系统，其中机器学习算法以高级语言表达并在 MapReduce 环境中执行。高级语言公开多个构造，这些构造组成广泛的监督和无监督机器学习算法的关键构件。以高级语言表达的算法被编辑并优化成一组在机器集群上运行的 MapReduce 作业。本公开还描述了多个优化策略，以便在 MapReduce 框架(例如，**Apache®** Hadoop!)上高效执行这些算法。

[0019] 如下面说明所描述的，用于编写 ML 算法的声明式高级语言使用户在 MapReduce 中实现算法时免于低级实现细节和性能调整。此外，所述系统和方法提供可扩展到超大型数据集的性能。这种性能堪比针对个体算法的手动调整的实现。

[0020] MapReduce

[0021] 在操作上，MapReduce 包括三个阶段：“映射”阶段，其中将输入数据分成不同的键 - 值对；“混洗(shuffle)”阶段，其中将来自不同映射器的相同键聚集在一起；以及“化简”阶段，其中共同分析与单个键关联的所有值。通常，“映射”和“化简”阶段是公开的，而“混洗”阶段在平台内部。但是，“混洗”阶段的成本是本公开中描述的优化的一个重要方面。

[0022] SystemML

[0023] 图 1 示出用于在 MapReduce 环境中实现 ML 算法的系统 101（在本公开中称为“SystemML”）的一个示例性实施例。SystemML101 总体上包括用于接受声明式机器学习语言(DML)脚本 103 的语言组件 102、耦合到语言组件 102 的高级运算符(HOP)组件 104、耦合到 HOP 组件 104 的低级运算符(LOP)组件 105，以及耦合到 LOP 组件 105 的运行时组件 106。在一个实施例中，运行时组件 106 与分布式网络或集群 107-108a-n 连接以执行脚本。

[0024] 声明式机器学习语言

[0025] 声明式机器学习语言(DML)是一种声明式语言，其语法与编程语言 R 的语法非常类似。为了实现更多的系统生成的优化，DML 并不提供 R 中可用的所有灵活性。但是，灵活性的丧失主要导致程序方便性的丧失，而不会太多地影响以 DML 表达 ML 算法类。使用实例脚本 1（在以下提供）说明 DML 构造，该脚本是高斯非负矩阵因子分解(GNMF)算法(算法 1，在以下提供)。

[0026] 算法 1 :GNMF

[0027] 1:V=read(“in/V”);

[0028] 2:W=read(“in/W”);

[0029] 3:H=read(“in/H”);

[0030] 4:max iteration=20;

[0031] 5:i=0;

[0032] 6:while i<max iteration do

[0033] 7:H=H*(W^T V/W^TWH);

[0034] 8:W=W*(V H^T/WHH^T);

[0035] 9:i=i+1;

[0036] 10:end while

[0037] 11:write(W, “out/W”);

[0038] 12:write(H, “out/H”);

[0039] 脚本 1 :GNMF

[0040] 1:V=readMM(“in/V”,rows=1e8,cols=1e5,nnzs=1e10);

[0041] 2:W=readMM(“in/W”,rows=1e8,cols=10);

[0042] 3:H=readMM(“in/H”,rows=10,cols=1e5);

[0043] 4:max iteration=20;

[0044] 5:i=0;

[0045] 6:while(i<max iteration){

[0046] 7:H=H*(t(W)%*%V)/(t(W)%*%W%*%H);

[0047] 8:W=W*(V%*%t(H))/(W%*%H%*%t(H));

[0048] 9:i=i+1;}

[0049] 10:writeMM(W, “out/W”);

[0050] 11:writeMM(H, “out/H”);

[0051] DML 支持三种主要数据类型 :矩阵、向量和标量。所支持的标量数据类型是整数、双精度、字符串和逻辑。矩阵或向量中的单元可以包括整数、双精度或字符串值。DML 程序包括一系列语句,且默认计算语义是个体语句的顺序求值。目前在 DML 中支持以下构造 :

[0052] ●输入 / 输出 :分别提供 ReadMM (矩阵乘法)和 WriteMM 语句,以便从文件中读取向量和矩阵并将向量和矩阵写入文件。可选地,在 Read 语句中,用户可以提供矩阵或向量的其它属性,例如维度和稀疏性(非 0 表项的数量或 nnzs)。

[0053] ●控制结构 :DML 中支持的控制结构包括 while 语句、for 语句和 if 语句。脚本 1 中的步骤 6-9 示出实例 while 语句。

[0054] ●赋值 :赋值语句包括表达式,并将表达式的结果赋予变量(例如,脚本 1 中的步骤 7-9)。可以为标量、向量或矩阵赋值。

[0055] DML 支持以下主要类型的运算符 :

[0056] ●算术 :算术运算符包括乘法、除法、加法和减法。算术运算符自然扩展到向量和矩阵,其中语义决定将运算符应用于对应的单元。

[0057] ●关系 / 布尔 :关系 / 布尔运算符包括小于、大于和等于。

[0058] ●内部函数：内部函数包括聚合函数(例如，sum、rowSum)、数学函数(例如 sin 和 log)，以及线性代数函数(例如，transpose t、diag)。

[0059] DML 还允许用户使用语法“函数(参数列表)主体”定义他们自己的函数。参数列表包括一组形式输入和输出参数。主体是一组有效的 DML 语句。

[0060] 将 DML 与 R 编程语言相比，DML 不支持高级编程特性，例如面向对象的特性、高级数据类型(例如，列表和数组)以及高级函数支持(例如，访问调用方函数中的变量，并进一步向上访问调用堆栈中的变量)。此外，DML 不支持 R 所支持的大量图形程序。

[0061] 用于 SystemML 的示例性方法

[0062] 图 2 示出由 SystemML101 在 MapReduce 环境中实现 ML 算法时执行的方法 200 的一个示例性实施例。此外，图 3 示出简单语句 A=B*(C/D) 的实例处理 300，其用于示例性地描述示例性方法 200。从 201 开始，SystemML101 的语言组件 102 接收用于要实现的 ML 算法的 DML 脚本 103。继续到 202，语言组件 102 将算法分解成较小单元(称为语句块)的解析表示。

[0063] 图 4 示出用于生成解析表示的方法 400 的一个示例性实施例。从 401 开始，为 DML 脚本中的每个变量指定数据类型。例如，从脚本 1，使用 ReadMM 语句(步骤 1-3)将 V、W 和 H 的类型指定为矩阵，而使用赋值语句(步骤 4-5)将 max iteration 和 i 标识为标量变量。

[0064] 继续到 402，将程序分成语句块。继续脚本 1 中的实例，将连续赋值、ReadMM 和 WriteMM 语句组合成单个语句块，因为可以共同优化这些语句中涉及的运算。控制结构(例如，while 循环和函数)引入自然边界，语句块不能跨越这些自然边界。脚本 1 分成三个语句块(在以下提供)：

[0065] 语句块 1

[0066] 1:V=readMM(“in/V”,rows=1e8,cols=1e5,nnzs=1e10);

[0067] 2:W=readMM(“in/W”,rows=1e8,cols=10);

[0068] 3:H=readMM(“in/H”,rows=10,cols=1e5);

[0069] 4:max iteration=20;

[0070] 5:i=0;

[0071] 语句块 2

[0072] 6:while(i<max iteration)do

[0073] 7:H=H*(t(W)%*%V)/(t(W)%*%W%*%H);

[0074] 8:W=W*(V%*%t(H))/(W%*%H%*%t(H));

[0075] 9:i=i+1;

[0076] 语句块 3

[0077] 10:writeMM(W,“out/W”);

[0078] 11:writeMM(H,“out/H”);

[0079] 在一个实施例中，存在不同类型的语句块。不同类型的语句块可以包括：(1)简单语句块(例如，包括在执行期间运行一次的基本脚本，例如上面的语句块 1)；(2)重复语句块(其脚本可以执行多次的语句块，例如，代码中的循环，包括 for、while、do-while 等，例如上面的语句块 2)；以及(3)跳过语句块(其脚本可能不执行的语句块，例如，条件语句，例如 if 等)。当执行语句块的关联 lop 时，语句块的类型可以用于确定如何以及是否执行低级运算

(例如,对于其 if 条件未被满足以执行的跳过语句块,跳过 lop)。

[0080] 继续到 403,SystemML101 确定需要跨语句块传递何种变量。例如,步骤 7 中使用的变量 W 引用步骤 2 的输出(对于循环的第一次迭代),并且对于向前的第二次迭代,引用步骤 8 的输出。在确定需要跨语句块传递何种变量时,脚本中变量的每次使用都与跨不同求值路径的该变量的紧接之前的写入(多个)关联。

[0081] 再次参考图 2,继续到 203,HOP 组件 104 确定多个语句块的高级执行计划。HOP 组件 104 接收语句块的解析表示作为输入,并通过矩阵、向量和标量生成基本运算(hop)的有向非循环图(HOPDag)。以下是 SystemML101 中的可用 hop (及其语义):

[0082] ●二元(例如,b(/)):二元 hop 采取两个输入(其中每个输入是矩阵、向量或标量)并执行以下运算之一 :*、+、-、/、min、max 等。

[0083] ●一元(例如,u(+)):一元 hop 采取两个操作数作为输入 :(1)矩阵、向量或标量,以及(2)标量常数。一元 hop 然后执行以下运算之一 :*、+、-、/。

[0084] ● AggUnary (例如,a(+)):AggUnary hop 聚合矩阵或向量中的一组单元。例如,colSum 计算矩阵中的每列的总和并返回总和的向量。

[0085] ● AggBinary (例如,a(+*)):AggBinary hop 采取两个输入,其中每个输入是矩阵或向量。AggBinary hop 然后执行二元运算,后跟聚合运算(例如,+、Min、Max 等)。

[0086] ● Reorg (例如,r(T)):reorg hop 更改矩阵中的每个单元的下标,例如矩阵的转置。

[0087] ●数据 :数据 hop 读取或写入矩阵、向量或标量。相对于脚本的生命周期,数据 hop 可以是临时的或持久的。

[0088] 在创建 HOPDag 时,以一个 HOPDag 表示语句块。图 5 示出使用上面针对脚本 1 的语句块 2 中的 While 循环主体引入的 hop 的 HOPDag。在一个实施例中,可以将语句块中的多个语句组合成单个 HOPDag。例如,读取 W 数据 hop 馈入 r T reorg hop 中,r T reorg hop 又与读取 W 数据 hop 一起馈入 a(+*)AggBinary hop 中以表示语句块的 t(W)%*%W 部分。解析并分析程序时,hop 被实例化并被连接以便针对程序中的语句块构建 HOPDag。尽管该图示出 HOPDag 被连接,但应该指出,HOPDag 不是必须被连接。

[0089] 灰色的读取数据 hop 表示迭代开始时的矩阵 W、H 和 V 以及标量 i 的实时输入的变量。灰色的写入数据 hop 表示迭代结束时的实时输出的变量,这些变量需要传递到下一次迭代。这些临时的数据 hop 隐式地将不同语句块的 HOPDag 彼此连接,方式是将一个语句块的 HOPDag 的临时写入数据 hop (宿)映射到下一个语句块或 while 循环的下一次迭代的 HOPDag 的临时读取数据 hop (源)。

[0090] 再次参考图 2,继续到 205,LOP 组件 105 将 HOPDag 转变成低级物理执行图(LOPDag)。低级运算符(lop)表示 MapReduce 环境中的基本运算。每个 lop 采取一组或多组键 - 值对作为输入,并生成一个或多个键 - 值对作为输出。以下是 SystemML101 中支持的 lop :

[0091] ●二元 :二元 lop 执行涉及两个输入的计算,其中每个输入是一组键 - 值对。

[0092] ●一元 :一元 lop 执行如下计算 :其中一个输入是一组键 - 值对,另一个输入是标量值。

[0093] ●分组 :分组 lop 对所有具有相同键的键 - 值对进行分组。

[0094] ●聚合 :聚合 lop 针对与相同键关联的一组值应用聚合函数。

[0095] ●变换 :变换 lop 针对每个输入键应用变换函数(例如,transpose)。

[0096] ●数据 :数据 lop 表示输入数据源或输出数据宿。

[0097] ●支持矩阵乘法的 lop :

[0098] ○ mmcj :mmcj lop 对应于基于叉积的矩阵乘法(CPMM)中的叉积计算。

[0099] ○ mmrj :mmrj lop 对应于基于复制的矩阵乘法(RMM)中的计算。

[0100] 在创建 LOPDag 的一个实施例中,以自下而上的方式处理 HOPDag,其中将每个 hop 转换成一个或多个 lop。图 6 示出二元 hop C/D (图 3 中所示,来自算法 1)的对应 lop600 的一个实例。在底部,两个数据 lop 中的每一个都针对输入矩阵返回一组键 - 值对。在概念上,将针对个体矩阵中的每个单元返回一个表项。但是,数据 lop 通常针对每个键返回多个单元(返回的单元数量由适当的分块策略确定)。分组 lop 然后对两个输入的多个表项进行分组。然后将结果传递到二元 lop 以便对来自两个输入矩阵的对应单元 - 值执行除法。

[0101] 在一个实施例中,可以在创建 LOPDag 时使用基于成本的优化。创建 LOPDag 时,可能存在多种用于将 hop 转变成一个或多个 lop 的选择。因此,可以使用考虑了所涉及的矩阵的各种数据特征的基于成本的优化,以便降低针对 hop 选择的 lop 组的事务成本。基于成本的优化的一个实例包括从执行矩阵乘法的多种方法中进行选择,如下所述。

[0102] 返回到图 2,继续到 205,LOP 组件 105 将 LOPDag 编辑成至少一个 MapReduce 作业。将每个 lop 转变成单独的 MapReduce 作业将导致多次扫描输入数据和中间结果。因此,如果将多个 lop 打包成单个 MapReduce 作业,则产生的扫描次数的减少通常导致效率提高。lop 存在多个属性,它们有助于将多个 lop 打包成一个 MapReduce 作业。例如,以下是两个此类属性 :

[0103] ●位置,表示可以在“映射”或“化简”或这两者中执行 lop。

[0104] ●键特征,表示应该对输入键进行分组,对生成的输出键进行分组,以及 lop 是否生成新的输出键。

[0105] 在一个实施例中,使用贪婪捎带(piggybacking)启发式算法(在下面为算法 2)分析多个 lop 并将它们分组成一个 MapReduce 作业。

[0106] 算法 2:捎带—将可以共同求值的 lop 打包在单个 MapReduce 作业中

[0107]

Input: LOP-Dag
Output: A workflow of MapReduce Jobs(MRJobs)
$[N_{Map}, N_{MapOrRed}, N_{MapAndRed}, N_{Red}]$ = TopologicalSort(LOP-Dag);
while(Nodes in LOP-Dag remain to be assigned) ***do***
  Job ←create a new MapReduce job;
  addNodesByLocation($N_{Map} \cup N_{MapOrRed}$, Map, Job);
  addNodesByLocation($N_{MapAndRed}$, MapAndReduce, Job);
  addNodesByLocation($N_{MapOrRed} \cup N_{MapAndRed} \cup N_{Red}$, Reduce, Job);
  add Job to MRJobs;
end while
return MRJobs

[0108]

```
{Method to add nodes that are ready to be assigned for a specific execution location}
Method: addNodesByLocation ( S, loc, Job )
while(true) do
  Z ← Φ
  while( S is not empty ) do
    n ←S.next()
    if(n is not yet assigned and all descendants of n have been assigned) then
      if(loc is Map ) then
        add n to Z
      else if( loc is MapAndReduce ) then
        add n to Z if n does not have any descendant lop in Z and Job whose location is
MapAndReduce
      else if(loc is Reduce) then
        if( n is a group lop ) then
          add n to Z only if n has a descendant group lop in Z or Job; and none of the lops
between these two group lops alter keys
        else
          add n to Z if n is not a group lop
        end if
      end if
    end if
  end while
  break if Z is empty
  add Z to Job.Map, Job.MapAndReduce, or Job.Reduce, based on loc
end while
```

[0109] 为了继续脚本 1 中的实例，图 7 示出针对步骤 7 将 lop 打包成 MapReduce 作业的一个实例 700。701（700 的左侧部分）示出标记为“H 赋值”的 HOPDag500（图 5）的部分的已生成 LOPDag。702（700 的右侧部分）示出 LOPDag701 的对应 MapReduce 作业。

[0110] 返回到图 2，继续到 206，运行时组件 106 在运行时优化 MapReduce 作业（多个）的执行。在一个实施例中，SystemML101 的运行时组件 106 存在三个主要考虑事项：（1）矩阵和向量的键 - 值表示；（2）通过 MapReduce 执行个体 LOPDag 的 MR 运行时；以及（3）用于协调执行的控制模块。

[0111] 对于矩阵和向量的键 - 值表示，SystemML101 将矩阵和向量分成块（称为分块），并利用块中的局部稀疏性来优化表示矩阵和向量的键 - 值对的数量。块是使用指定的块大小的较小矩形子矩阵。每个块以键 - 值对表示。键表示块 id。值包含块中的所有单元值。局部稀疏性指个体块的稀疏性。根据块中的稀疏性（即，块中的非 0 值部分）确定块中的值布局。

[0112] 在一个实施例中，动态块级别运算基于块的局部稀疏性。因此，局部稀疏性信息用于确定每个块在运行时的适当执行。在一个实施例中，每个 lop 内部都具有单独算法以便考虑个体块可能稠密或稀疏的事实。

[0113] 例如，如果要针对两个个体块执行矩阵乘法，则根据两个输入块的局部稀疏性确定在 lop 中选择的实际乘法算法。如果两个块均稠密，则运行时选择遍历两个块中的每一

个单元的算法。但是，如果一个块稀疏，则运行时选择仅遍历稀疏块中的非零单元的算法，这些非零单元与稠密块中的对应单元中的值相乘。

[0114] 对于通过 MapReduce 执行个体 LOPDag 的 MR 运行时，通用 MapReduce 作业（G-MR）是 SystemML101 中的主要执行引擎。由捎带算法（上面的算法 2）使用一个或多个 lop 实例化 G-MR。为了示出 G-MR 的示例实例化，图 7 的 702 中标记为 1 的 MapReduce 作业包括三个 lop：数据 W；变换；以及 mmcj。为了实例化 MapReduce 作业，按如下方式参数化 G-MR：

[0115] ● MapReduce 作业的“映射”阶段按照 LOP 组件 105 指示，顺序运行数据、变换以及 mmcj 的映射部分。

[0116] ● MapReduce 作业的“化简”阶段执行 mmcj 的化简部分。

[0117] 运行时组件 106 的控制模块针对 DML 脚本协调所有 MapReduce 作业的执行。在一个实施例中，控制模块执行以下操作：（i）指令调度以及（ii）指令执行。在 DML 脚本执行期间，在控制模块中执行的此类操作包括脚本中的标量计算（例如，标量算术运算和谓词求值）和元数据操作（例如，删除中间结果）。

[0118] SystemML101 可以执行得到的 MapReduce 作业，方式是将得到的作业发送到用于在分布式网络 107 上运行应用的框架。一个实例框架是用于在分布式节点 108a-n 上处理作业的**Apache®** Hadoop!。

[0119] 矩阵乘法

[0120] SystemML 101 支持至少两种矩阵乘法算法：RMM 和 CPMM。为了示出 RMM 和 CPMM，假设 A 和 B 是分块矩阵，A 中具有 $M_b \times K_b$ 个块，B 中具有 $K_b \times N_b$ 个块。块级别处的矩阵乘法计算对应于 $C_{i,j} = \sum_k A_{i,k} \times B_{k,j}$，其中下标表示块 id。

[0121] 图 8 示出 RMM 算法 800，其仅需要一个 MapReduce 作业以便执行。包括 RMM 的执行计划的 LOPDag 包含单个 mmrj lop（在 G-MR 中实现）。在 RMM 算法 800 中，化简器用于访问 A 和 B 的所有块，这些块是计算 C 的每个结果块必需的。因为每个块 A 均有助于若干结果块的计算，所以映射器将用于计算若干结果块的 A 的副本数量发送到对应的化简器。也针对每个块 B 执行相同的操作，化简器现在具有用于计算 C 所需的所有块。

[0122] 图 9 示出 CPMM 算法 900，其需要两个 MapReduce 作业以便执行。CPMM 在 LOPDag 中表示为顺序的三个 lop：mmcj；分组；以及聚合。再次参考图 7，实例 700 示出用于计算 W^TW 的 CPMM 求值。第一 MapReduce 作业的映射器读取两个输入矩阵 A 和 B，并根据通用键 k 对来自 A 和 B 的输入块进行分组。因此，化简器执行叉积以计算 C（$C^k_{i,j} = A_{i,k} \times B_{k,j}$）。在第二 MapReduce 作业中，映射器从前一个 MapReduce 作业读取结果，并根据键 (i, j) 对所有 $C^k_{i,j}$ 进行分组。在“化简”阶段中，聚合 lop 计算 $C_{i,j} = \sum_k C^k_{i,j}$。

[0123] 对于 CPMM，SystemML 101 可以包括 mmcj 的优化实现。在一个实施例中，优化实现是在化简器中实现部分聚合的局部聚合器。第一 MapReduce 输出是 $C^k_{i,j}$（对于 $1 \leqslant k \leqslant K_b$）。当 K_b 大于可用化简器的数量 r 时，每个化简器可以处理多个组。例如，化简器可以针对

k=k’ 和 k=k’’ 应用叉积，然后同一化简器将计算 $C^{k'}_{i,j}=A_{i,k'}\times B_{k',j}$ 和 $C^{k''}_{i,j}=A_{i,k''}\times B_{k'',j}$。如前面针对 CPMM 所述，来自第一 MapReduce 作业的输出在第二 MapReduce 作业中聚合为 $C_{i,j}=\sum_k C^k_{i,j}$。因此，局部聚合器可以在化简器中进行部分聚合，而不是单独输出 $C^{k'}_{i,j}$ 和 $C^{k''}_{i,j}$。

[0124] 在一个实施例中，为了防止部分聚合太大以至于无法纳入存储器中，可以实现基于磁盘的局部聚合器。基于磁盘的局部聚合器被配置为使用存储器中的缓冲池以执行局部聚合。如果叉积结果溢出到磁盘，则可以对结果进行排序以确保高效执行后续组的部分聚合。

[0125] 对于矩阵乘法，SystemML101 在 CPMM 和 RMM 之间进行选择。在一个实施例中，SystemML101 通过比较使用不同算法的成本模型来优化选择。对于 RMM，映射器复制 A 和 B 的每个块，次数等于要针对 A 和 B 的每个块计算的 C 的块总数（对于 A 记录为数量 N_b，对于 B 记录为数量 M_b）。因此，在 MapReduce 作业中混洗 $N_b|A|+M_b|B|$ 数据。因此，RMM 的成本模型是 $cost(RMM)=shuffle(N_b|A|+M_b|B|)+IO_{dfs}(|A|+|B|+|C|)$。

[0126] 对于 CPMM，在第一 MapReduce 作业中，映射器读取 A 和 B 的块，并将这些块发送到化简器。因此，混洗的数据量是 |A|+|B|。如前所述，化简器针对每个 k 执行叉积，并应用局部聚合器以便跨化简器中的不同 k 值部分地聚合结果。因此，每个化简器生成的结果集的大小通过 |C| 界定。因此，当作业中具有 r 个化简器时，写入 DFS 的数据量通过 r|C| 界定。在第二 MapReduce 作业中，来自第一 MapReduce 作业的数据被读取、混洗并馈入化简器中以生成最终结果。因此，CPMM 的成本通过以下成本模型界定：$cost(CPMM)\leqslant shuffle(|A|+|B|+r|C|)+IO_{dfs}(|A|+|B|+|C|+2r|C|)$。

[0127] 因此，在一个实施例中，SystemML101 将 cost(RMM) 与 cost(CPMM) 相比较，以便针对特定矩阵乘法确定适当的算法。在一个实例中，当 A 和 B 均非常大时，CPMM 的执行通常好于 RMM（因为 RMM 的混洗开销较大）。在另一个实例中，如果一个矩阵足够小而可以纳入一个块中，则开销足够低，使得 RMM 的执行通常好于 CPMM。应该指出，当数据混洗和 IO_{dfs} 操作具有相同大小时，数据混洗是更昂贵的操作，因为它涉及网络开销以及局部文件系统 IO 和外部排序。

[0128] 用于实现系统和方法的示例性计算机架构

[0129] 图 10 示出用于实现本公开中描述的系统和方法的一个实例计算机架构。图 10 的示例性计算系统包括：1）一个或多个处理器 1001；2）存储器控制集线器（MCH）1002；3）系统存储器 1003（其存在不同类型，例如 DDR RAM、EDO RAM 等）；4）高速缓存 1004；5）I/O 控制集线器（ICH）1005；6）图形处理器 1006；7）显示器 / 屏幕 1007（其存在不同类型，例如阴极射线管（CRT）、薄膜晶体管（TFT）、液晶显示器（LCD）、DPL 等）；以及 8）一个或多个 I/O 设备 1008。

[0130] 一个或多个处理器 1001 执行指令，以便执行计算系统实现的任何软件例程。指令通常包括针对数据执行的某种操作。数据和指令都在系统存储器 1003 和高速缓存 1004 中排序。高速缓存 1004 通常被设计为具有短于系统存储器 1003 的延迟时间。例如，高速缓存 1004 可能与处理器（多个）集成到相同硅芯片（多个）上和 / 或使用较快的 SRAM 单元构建，

而系统存储器 1003 可能使用较慢的 DRAM 单元构建。通过倾向于将更常用的指令和数据存储在高速缓存 1004 中而不是存储到系统存储器 1003，计算系统的整体执行效率将提高。

[0131] 谨慎地将系统存储器 1003 用于计算系统中的其它组件。例如，从计算系统的各种接口（例如，键盘和鼠标、打印机端口、LAN 端口、调制解调器端口等）接收的数据或者从计算系统的内部存储元件（例如，硬盘驱动器）取回的数据通常临时在系统存储器 1003 中排队，然后在软件程序的实现中由一个或多个处理器 1001 处理。同样，软件程序确定应通过计算系统接口之一从计算系统发送到外部实体的数据或者存储到内部存储元件的数据通常临时在系统存储器 1003 中排队，然后进行传输或排序。

[0132] ICH1005 负责确保在系统存储器 1003 及其适当的对应计算系统接口（以及内部存储器件，如果计算系统如此设计）之间正确地传递此类数据。MCH1002 负责在处理器（多个）1001、接口和内部存储元件之间管理对系统存储器 1003 访问的各种争先请求，这些请求的出现时间可能彼此相近。

[0133] 还在典型的计算系统中实现一个或多个 I/O 设备 1008。I/O 设备通常负责将数据传输到计算系统和 / 或从计算系统传输数据（例如，网络适配器）；或者，负责在计算系统中进行大规模非易失性存储（例如，硬盘驱动器）。ICH1005 在它本身和观察到的 I/O 设备 1008 之间具有双向点到点链路。

[0134] 所要求保护的系统的不同实施例的组件可以包括软件、硬件、固件或它们的任意组合。组件可以是软件程序，这些程序可用于运行专用或公用软件的公用或专用或通用处理器。软件还可以是专门针对签名创建和组织以及重新编辑管理而编写的专用程序。例如，所述系统的存储器可以包括但不限于硬件（例如软盘、光盘、CD-ROM 和磁光盘、ROM、RAM、EPROM、EEPROM、闪存、磁卡或光卡、传播介质或其它类型的介质 / 机器可读介质）、软件（例如，需要在硬件存储单元上存储信息的指令）或它们的任意组合。

[0135] 此外，本发明的元素还可以提供为机器可读介质以便存储机器可执行指令。机器可读介质可以包括但不限于适合存储电子指令的软盘、光盘、CD-ROM 和磁光盘、ROM、RAM、EPROM、EEPROM、闪存、磁卡或光卡、传播介质或其它类型的介质 / 机器可读介质。

[0136] 对于图 2 和 4 中所示的示例性方法，本发明的实施例可以包括如上所述的各种过程。所述过程可以包含在机器可执行指令中，这些指令导致通用或专用处理器执行特定步骤。备选地，这些过程可以由包含用于执行所述过程的硬连线逻辑的特定硬件组件来执行，或者由编程计算机组件和定制硬件组件的任意组合来执行。

[0137] 本发明的实施例不需要提供的所有各种过程，并且本领域的技术人员可以构想如何在没有提供的特定过程的情况下实现本发明的实施例，或者使用未提供的额外过程实现本发明的实施例。例如，尽管在图 10 中描述了一个机器，但本公开的实施例可以跨多个机器实现。例如，可以在分布式计算环境中执行变换，其中可以在位于分布式网络中的不同计算机上执行所述方法的各部分。此外并且如图 1 中所示，可以在分布式计算环境中完成作业的执行。

[0138] 综述

[0139] 仅出于示例和说明目的给出上面对本发明的实施例的描述，并且所述描述并非旨在是穷举的或是将本发明限于所公开的精确形式。对于本领域的技术人员来说，在不偏离本发明的精神和范围的情况下，许多修改和改变都是显而易见的。

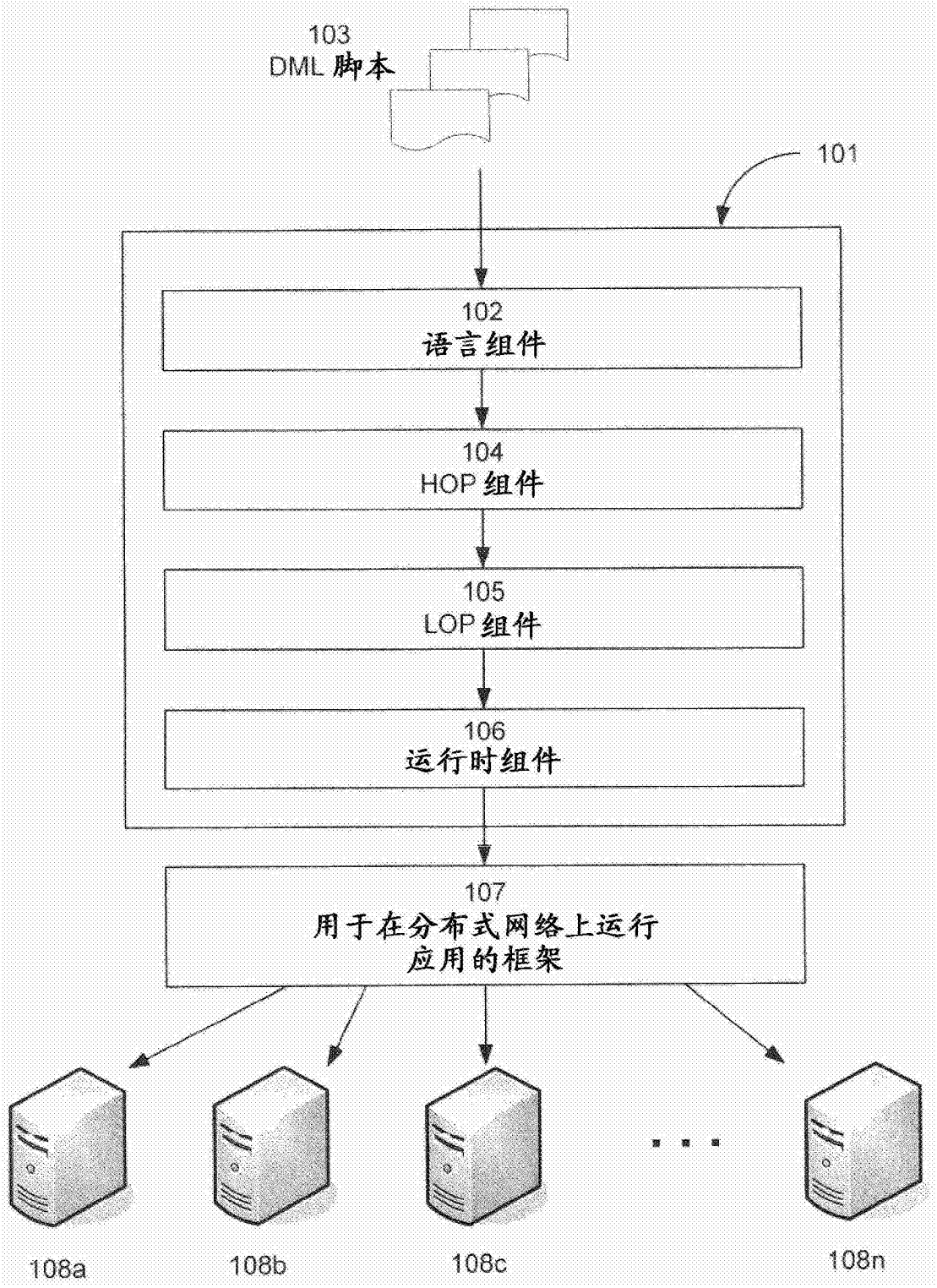

103
DML 脚本
101
102
语言组件
104
HOP 组件
105
LOP 组件
106
运行时组件
107
用于在分布式网络上运行
应用的框架
108a
108b
108c
108n

图 1

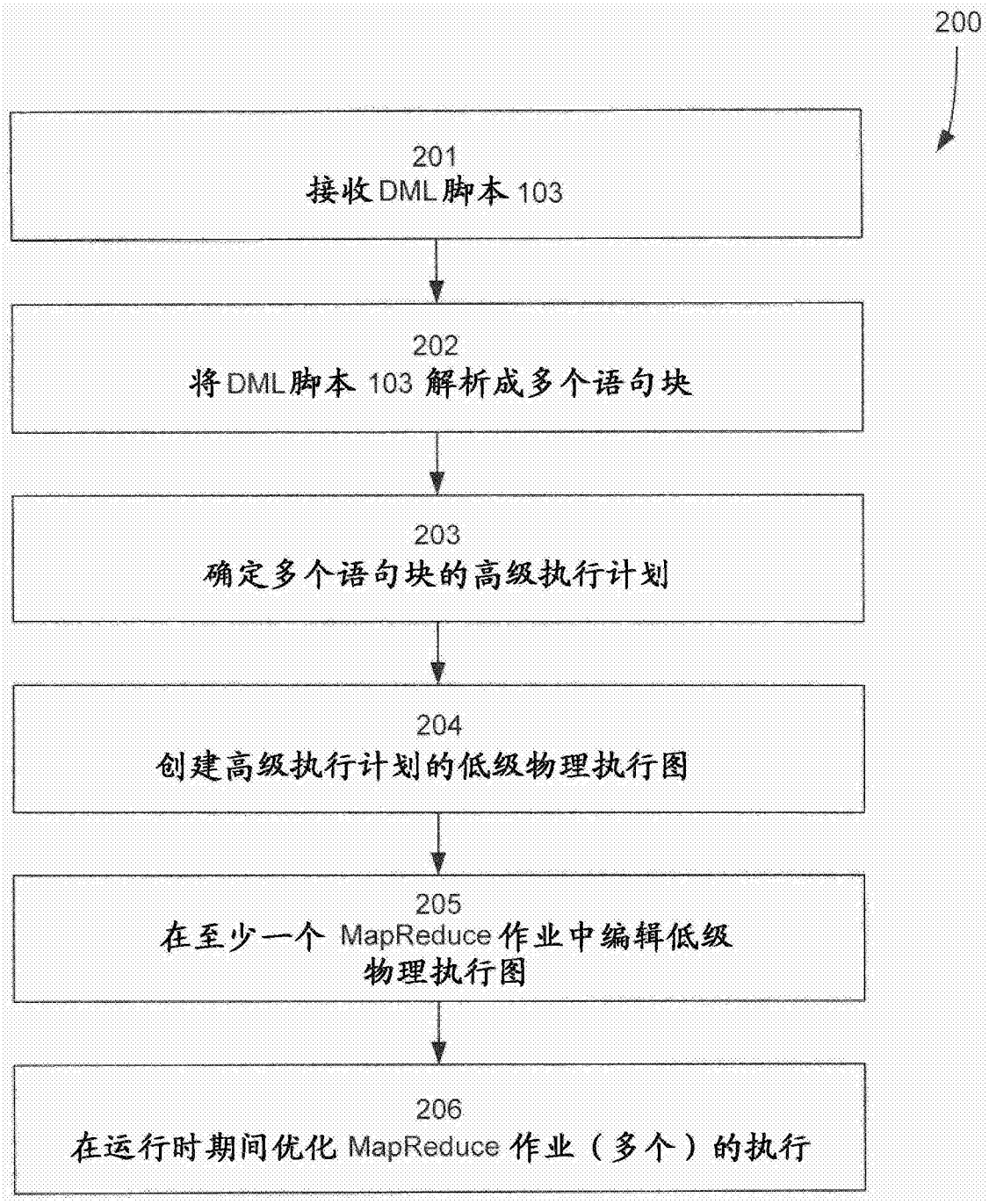

200
201
接收DML脚本103
202
将DML脚本103解析成多个语句块
203
确定多个语句块的高级执行计划
204
创建高级执行计划的低级物理执行图
205
在至少一个MapReduce作业中编辑低级物理执行图
206
在运行时期间优化MapReduce作业（多个）的执行

图 2

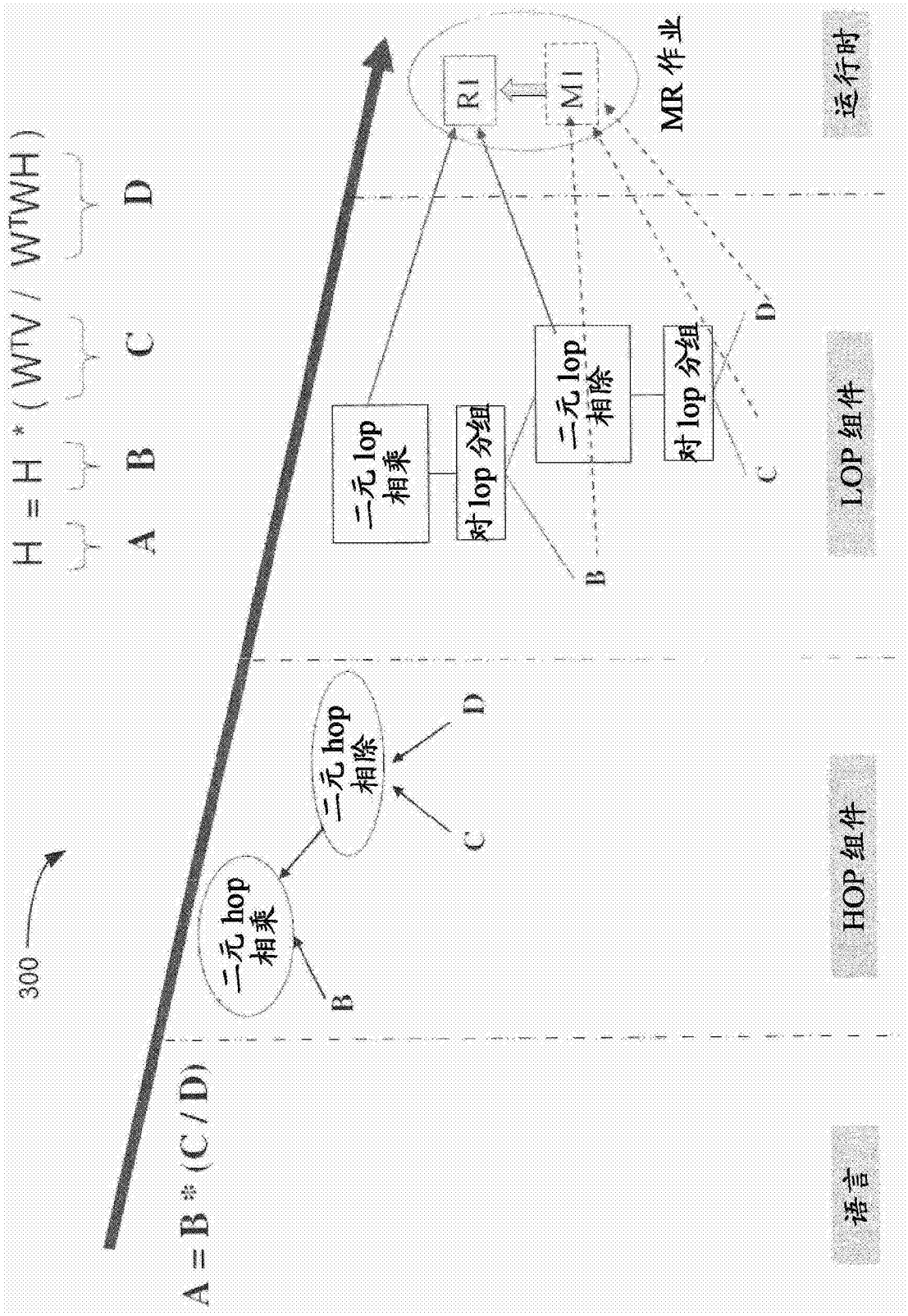

A = B * (C / D)
H = H * (WTV / WTWH)
A B C D
300
二元 hop 相乘
二元 hop 相除
B
C
D
二元 lop 相乘
对 lop 分组
二元 lop 相除
对 lop 分组
R1
M1
MR 作业
运行时
LOP 组件
HOP 组件
语言

图 3

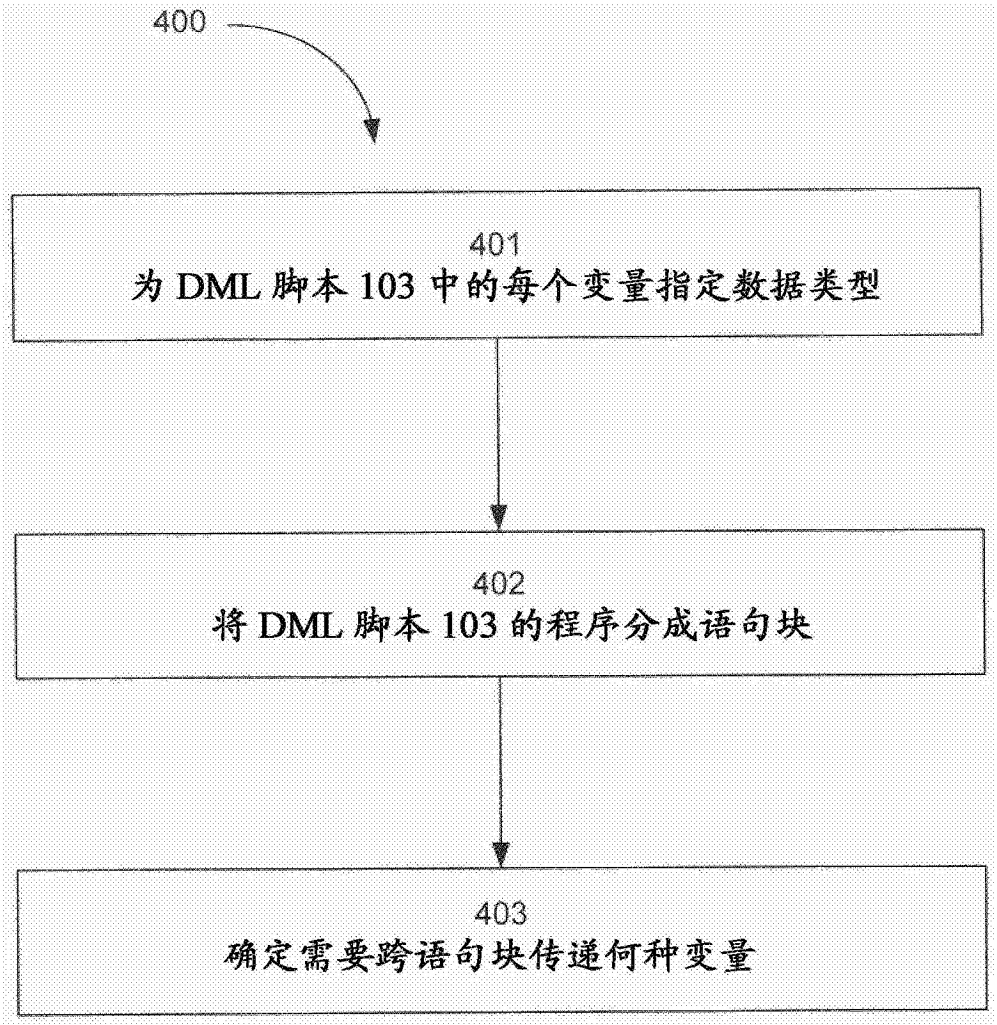

400
401
为 DML 脚本 103 中的每个变量指定数据类型
402
将 DML 脚本 103 的程序分成语句块
403
确定需要跨语句块传递何种变量

图 4

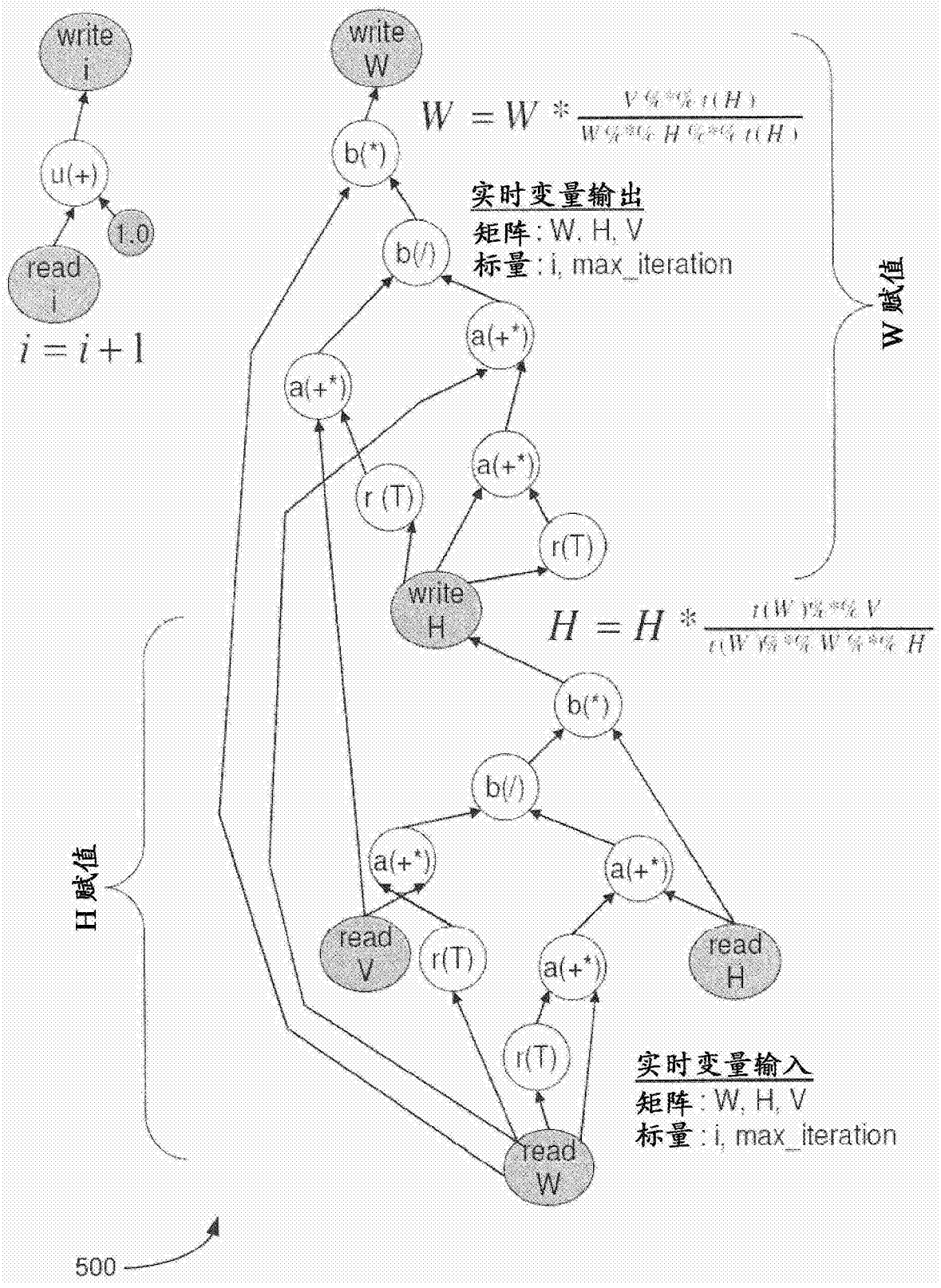

write i
u(+)
1.0
read i
$i = i + 1$
write W
b(*)
b(/)
a(+*)
a(+*)
a(+*)
r (T)
r(T)
write H
b(*)
b(/)
a(+*)
a(+*)
read V
r(T)
a(+*)
read H
r(T)
read W
$W = W * \frac{V^T * H}{W^T * H * H^T}$
$H = H * \frac{W^T * V}{W^T * W * H}$
W 赋值
H 赋值
实时变量输出
矩阵 : W, H, V
标量 : i, max_iteration
实时变量输入
矩阵 : W, H, V
标量 : i, max_iteration
500

图 5

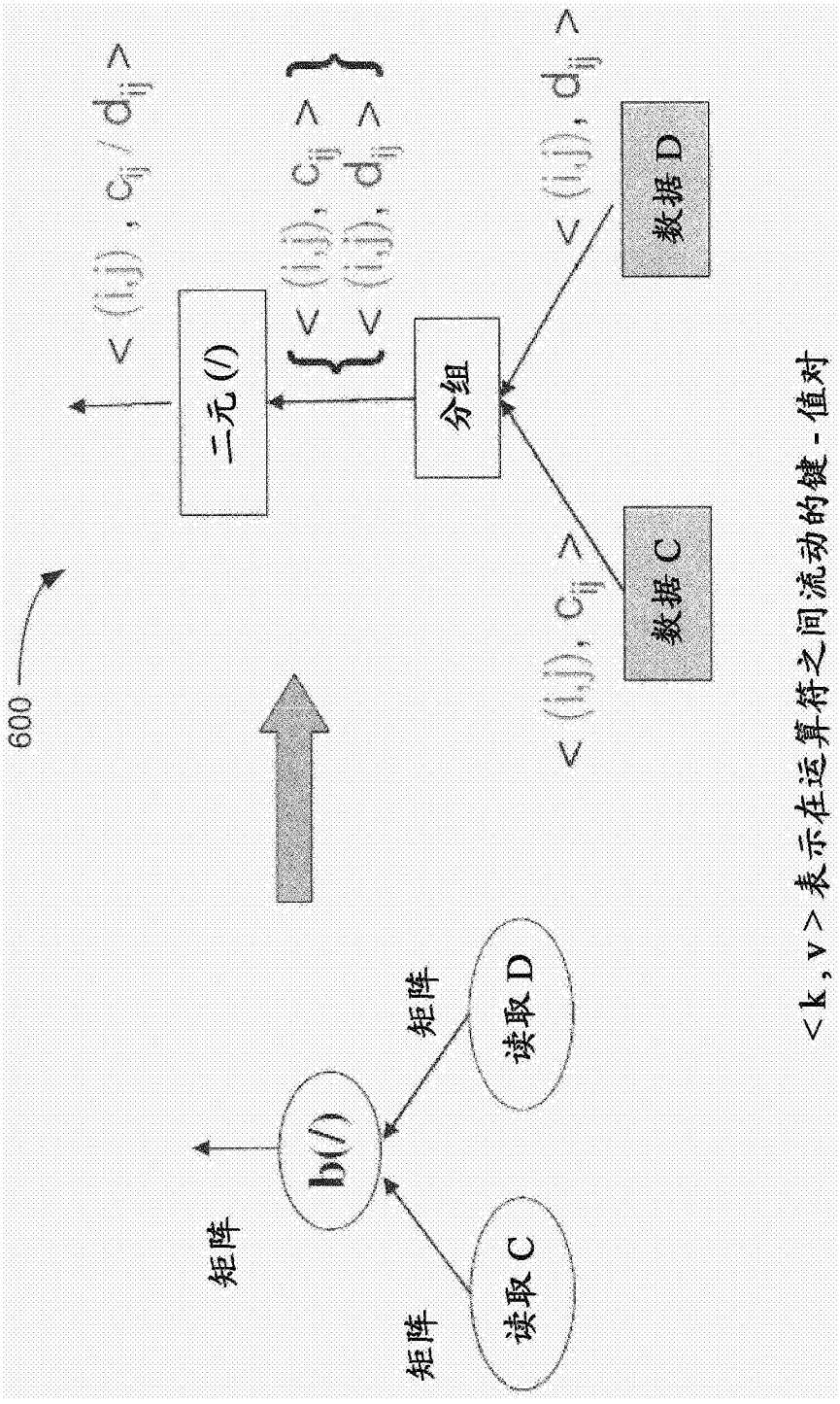


图 6

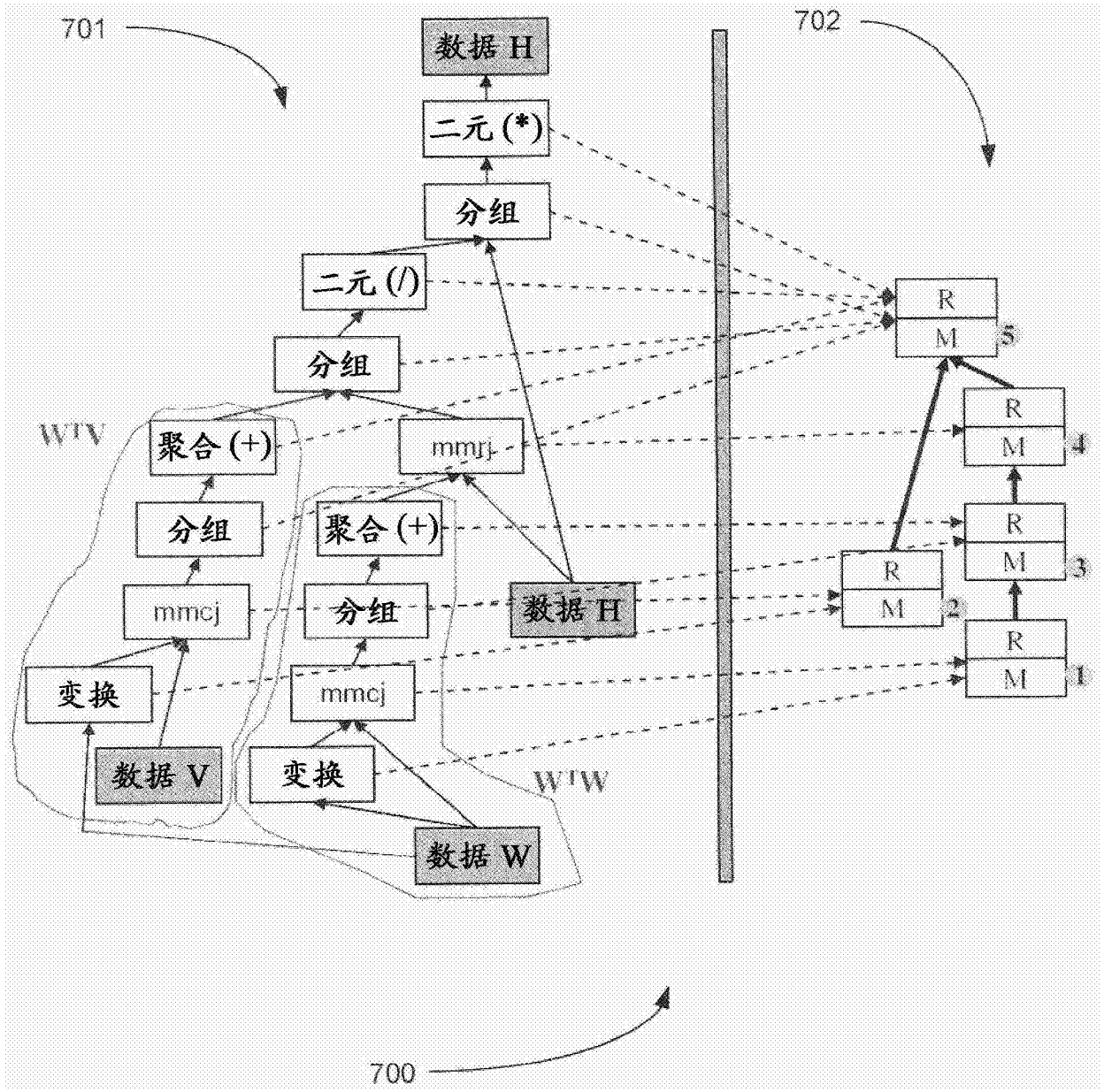

701
702
数据 H
二元 (*)
分组
二元 (/)
分组
聚合 (+)
mmrj
分组
聚合 (+)
mmcj
分组
数据 H
变换
mmcj
数据 V
变换
数据 W
W'V
W'W
R
M
5
R
M
4
R
M
3
R
M
2
R
M
1
700

图 7

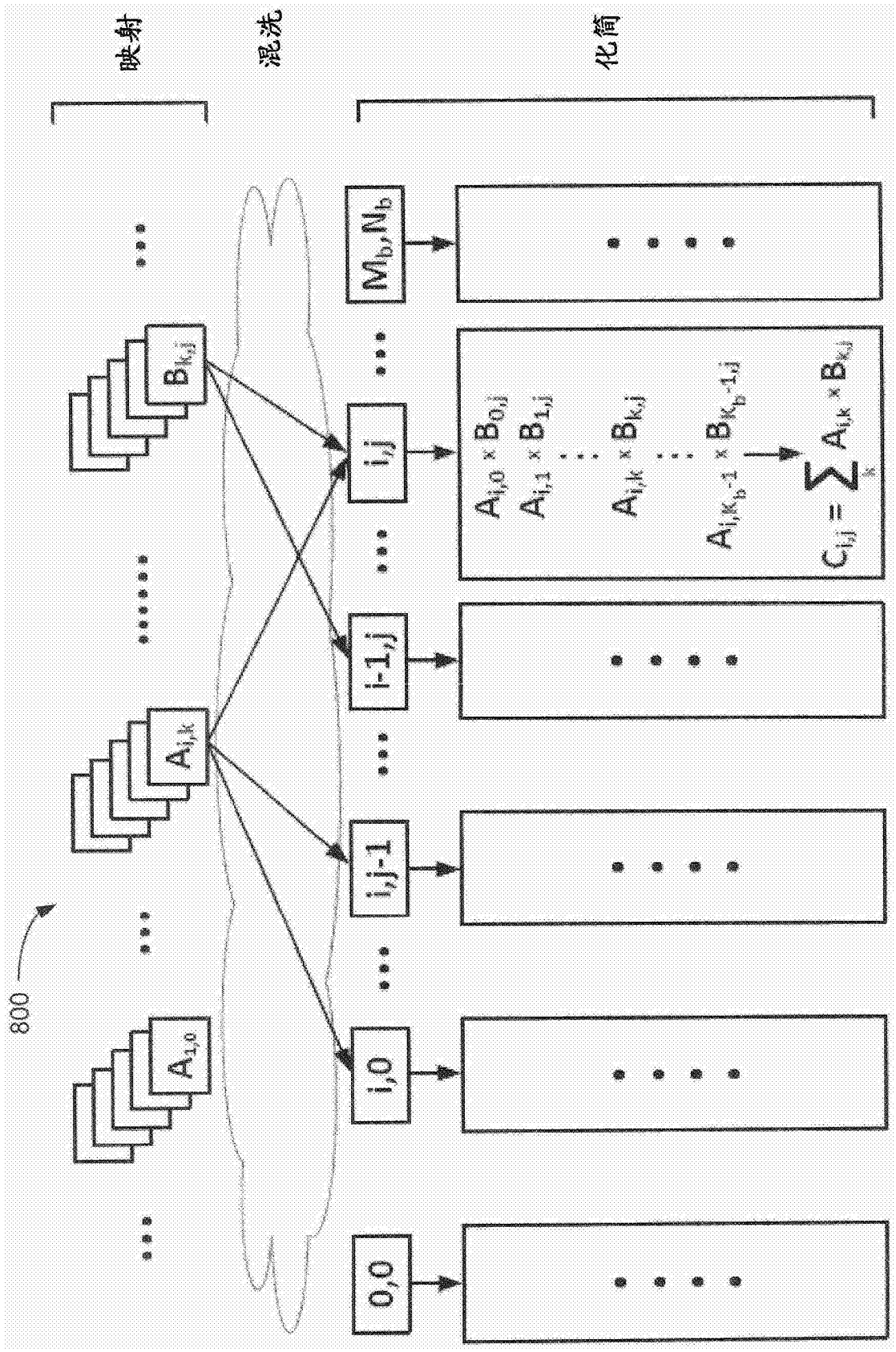

映射
混洗
化简
800
$A_{1,0}$
$A_{i,k}$
$B_{k,j}$
0,0
i,0
i,j-1
i-1,j
i,j
M_b, N_b
$A_{i,0} \times B_{0,j}$
$A_{i,1} \times B_{1,j}$
$A_{i,k} \times B_{k,j}$
$A_{i,K_b-1} \times B_{K_b-1,j}$
$C_{i,j} = \sum_k A_{i,k} \times B_{k,j}$

图 8

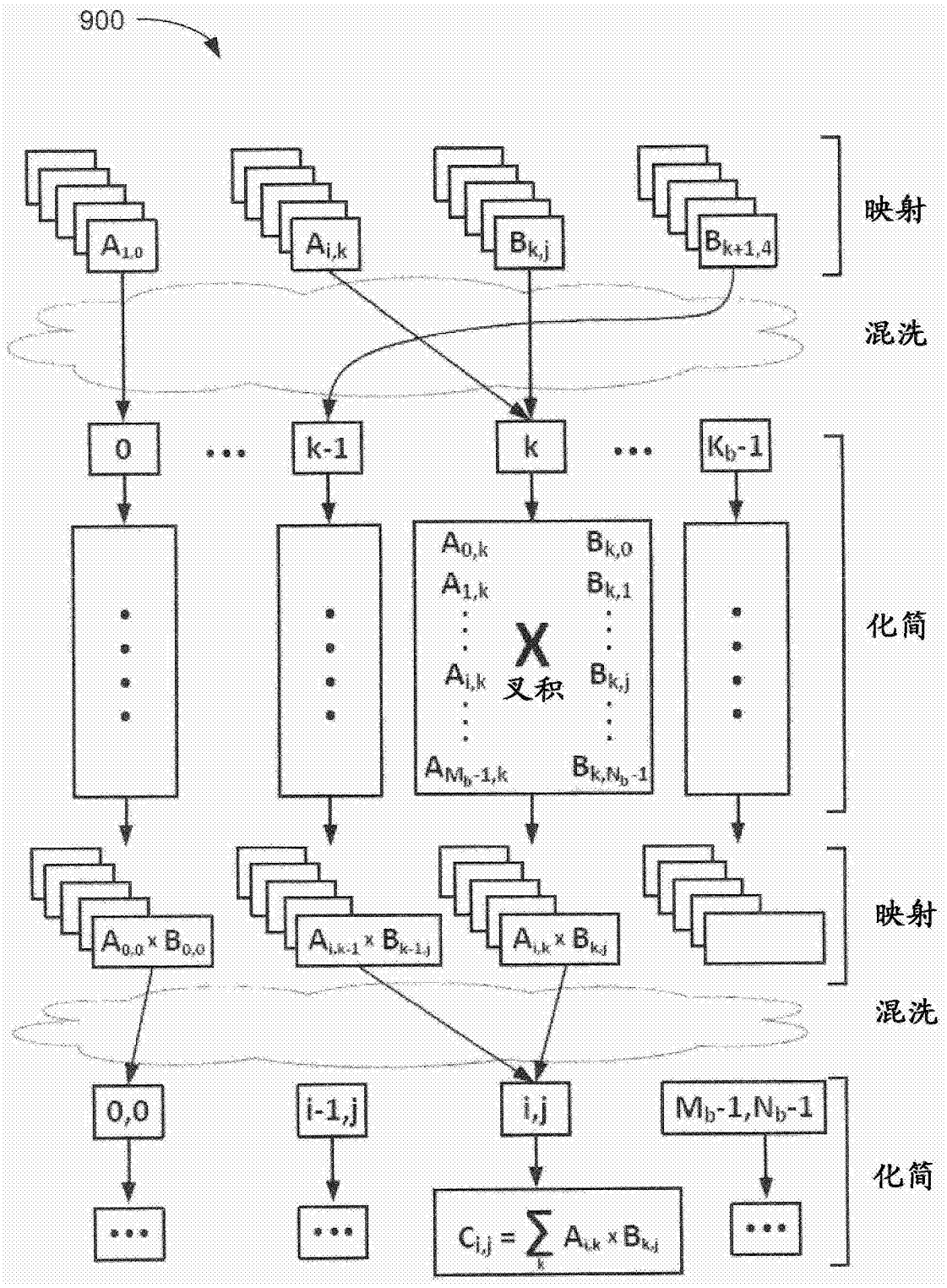


图 9

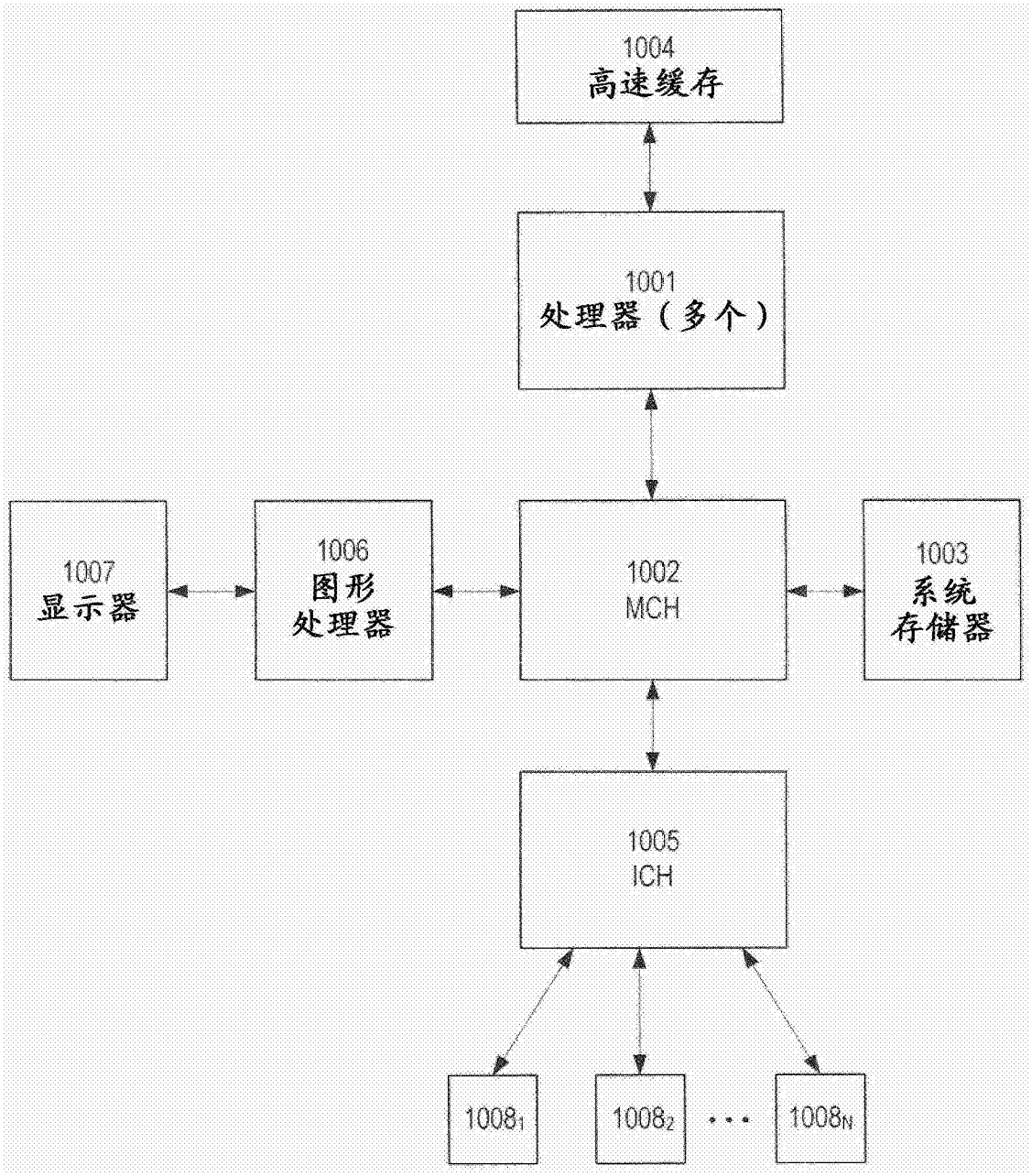

1004
高速缓存
1001
处理器（多个）
1007
显示器
1006
图形
处理器
1002
MCH
1003
系统
存储器
1005
ICH
1008₁
1008₂
1008N

图 10