利用Hadoop实现超大矩阵相乘之我见（二）

前文

在《利用Hadoop实现超大矩阵相乘之我见（一）》中我们所介绍的方法有着“计算过程中文件占用存储空间大”这个缺陷，本文中我们着重解决这个问题。

矩阵相乘计算思想

传统的矩阵相乘方法为行、列相乘的方式，即利用左矩阵的一行乘以右矩阵的一列。不过该方法针对稀疏矩阵相乘，会造成过多的无效计算，降低计算效率。为了解决这个问题，本发明采用列、行相乘计算方式，即利用左矩阵的一列中的元素与右矩阵对应行中的所有元素依次相乘，该方法有效避免了稀疏矩阵相乘过程中产生的无效计算。具体计算过程示意图如图1所示。

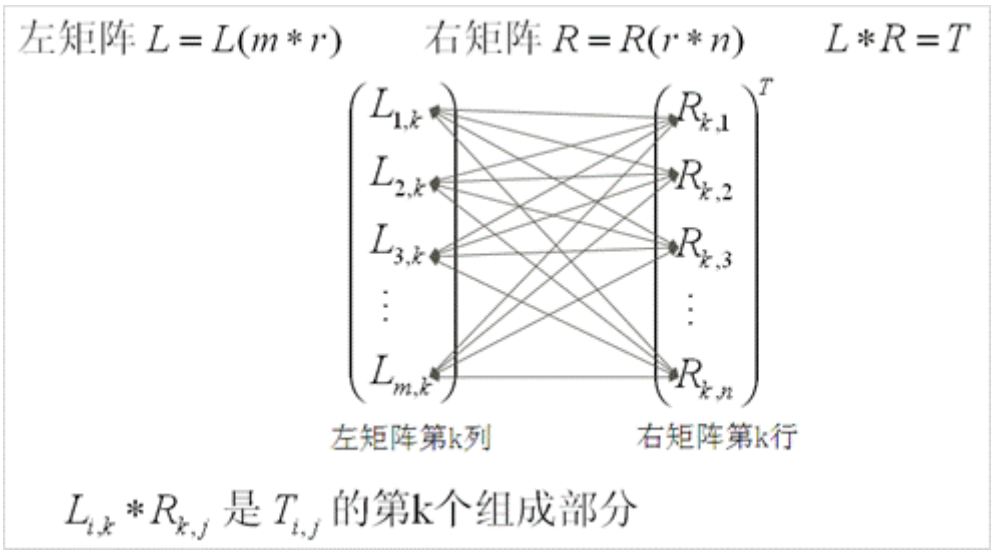

图1 列、行矩阵相乘计算示意图

数据预处理

为了便于Map-Reduce模型对矩阵元素进行处理，所有的矩阵元素都存储在文本文件中，一行记录代表一个矩阵元素，针对稀疏矩阵，0元素不纳入输入文本。如图2所示。

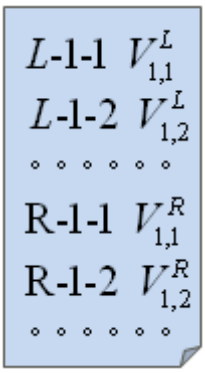

图2 输入的矩阵元素

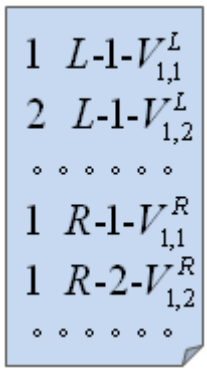

图3 预处理后的矩阵元素

我们对图2进行举例说明，假如一行记录为：$L\text{-}1\text{-}2\ V^L_{1,2}$。则其代表左矩阵第一行第二列的元素值为$V^L_{1,2}$。

在一个Map过程中，我们对每一行输入数据进行预处理，若一行记录代表左矩阵元素，则提取列号作为Key值，剩余信息组成Value值；若一行记录代表右矩阵元素，则提取行号作为Key值，剩余信息组成Value值，如图3所示。之所以这样做，是为了下一步在Reduce过程中能够按照Key值统计左矩阵第Key列极其对应右矩阵第Key行中的元素。

统计与分段

当矩阵规模大到一定程度时，内存可能会碰到加载不了左矩阵的一列或右矩阵的一行元素的问题。为了提高矩阵相乘运算的可扩展性，本发明提出了对左矩阵元素按列进行分段，对右矩阵元素按行进行分段的方法，这样，单个计算节点就可以加载左矩阵的一段与右矩阵的一段至内存进行相乘运算，突破了内存的限制。分段相乘示意图如图4所示。

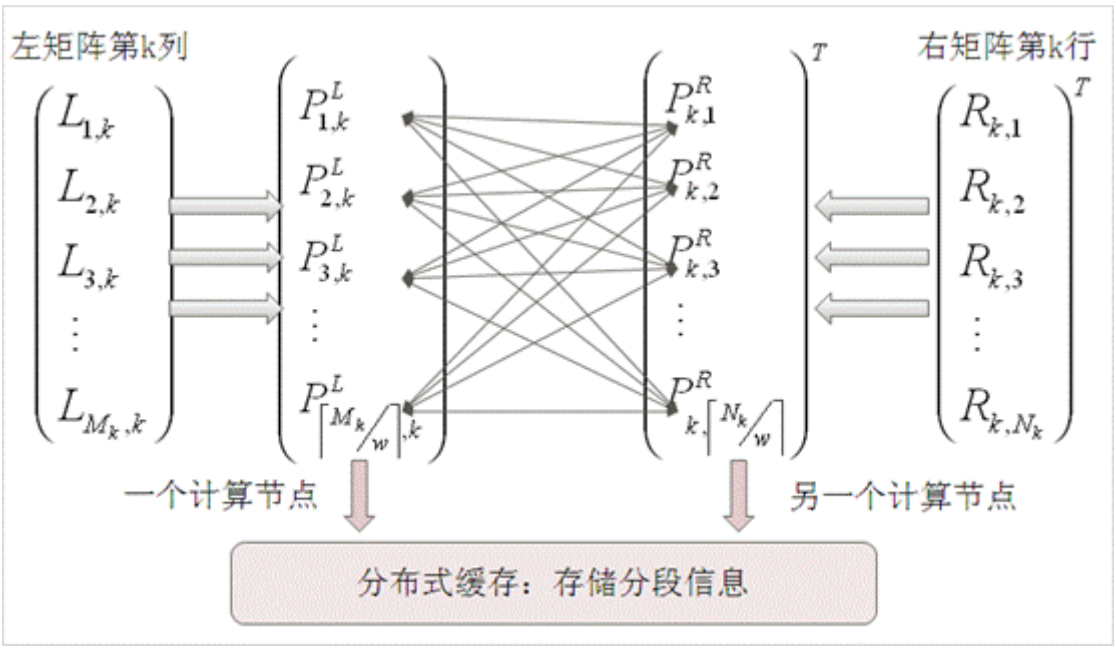

图4 矩阵分段相乘示意图

接下来我们结合图4来说明Reduce阶段如何来完成统计与分段工作。Reduce阶段首先将所有Key相同的Value集合在一起，形成一个Value-List。若Key为k，那么Value-List则代表了左矩阵第k列与右矩阵第k行的所有元素，这些元素时混合在一起的。在Reduce阶段，我们第一轮遍历Value-List,获得左矩阵第k列的元素个数为M_k，右矩阵第k行的元素个数为N_k。接下来我们通过第二轮遍历对左矩阵第k列、右矩阵第k行的元素进行分段操作，假设每个分段包含w个元素，则左矩阵第k列被分为段，右矩阵被分为段。

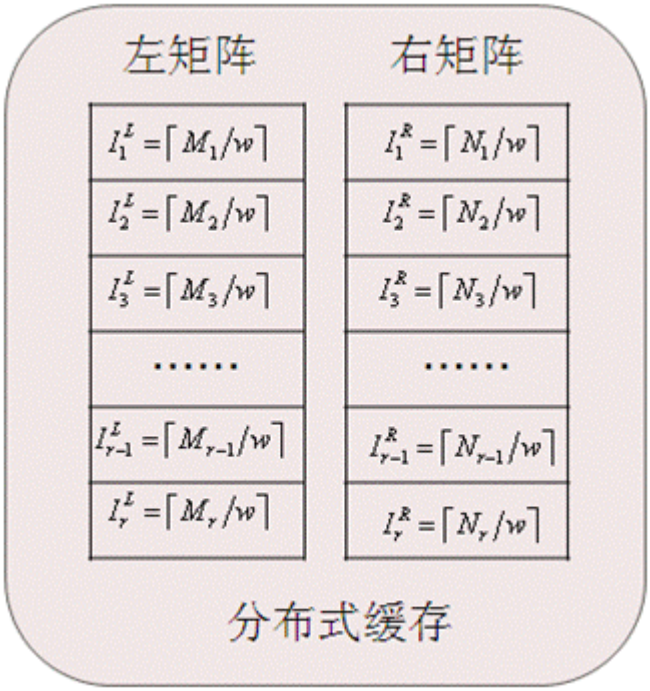

图5 分布式缓存存储矩阵分段信息

本发明将L矩阵中第k列中第i个分段表示成如下格式：

$$\text{k} \quad L-i-\lceil N_k/w\rceil-element\_list \quad i\in(1,2,\ldots,\lceil M_k/w\rceil) \qquad (1)$$

$\lceil N_k/w\rceil$代表该分段在接下来的过程中总共需要$\lceil N_k/w\rceil$个拷贝，element_list表示该分段中的元素集合。

同理，R矩阵中第k行中第j个分段表示成如下格式：

$$\text{k} \quad R-\lceil M_k/w\rceil-j-element\_list \quad i\in(1,2,\ldots,\lceil N_k/w\rceil) \qquad (2)$$

为了便于后续Map-Reduce过程的处理，我们将每一个分段都存储在磁盘文件中，文件中的一样代表一个分段。同时，我们将两个矩阵中的具体分段信息存储在分布式缓存中，有利于解决后续步骤中不同节点间的通信与数据查询问题。具体存储格式如图5所示。

图5中$I_1^L=\lceil M_1/w\rceil$代表矩阵L中第1列的元素个数为M_1，每个分段的元素个数为w，所以对该列的分段数目为I_1^L；同理$I_1^R=\lceil N_1/w\rceil$表示矩阵R中第1行的元素个数为N_1，每个分段的元素个数为w，所以对该行的分段数目为I_1^R。

拷贝任务分发——Map迭代算法

Map迭代算法

如图4所示，我们需要将两个矩阵中的分段一一对应相乘。我们做如下举例：由于矩阵L中的第k列中的第i段需要与矩阵R中第k行中的所有段依次相乘，所以需要将L中第k列第i段的内容拷贝$\lceil N_1/w\rceil$份；同理，R中第k行中的每一个分段需要拷贝$\lceil M_1/w\rceil$份。当然，拷贝工作是通过Map-Reduce来完成的，现在的问题是，若两个矩阵中每一个分段需要拷贝的数量都很大，则一个Map对每行记录都需要执行好多遍拷贝工作，大大延长了Map执行的时间，同时，可能使得很多计算节点没有参与运算。

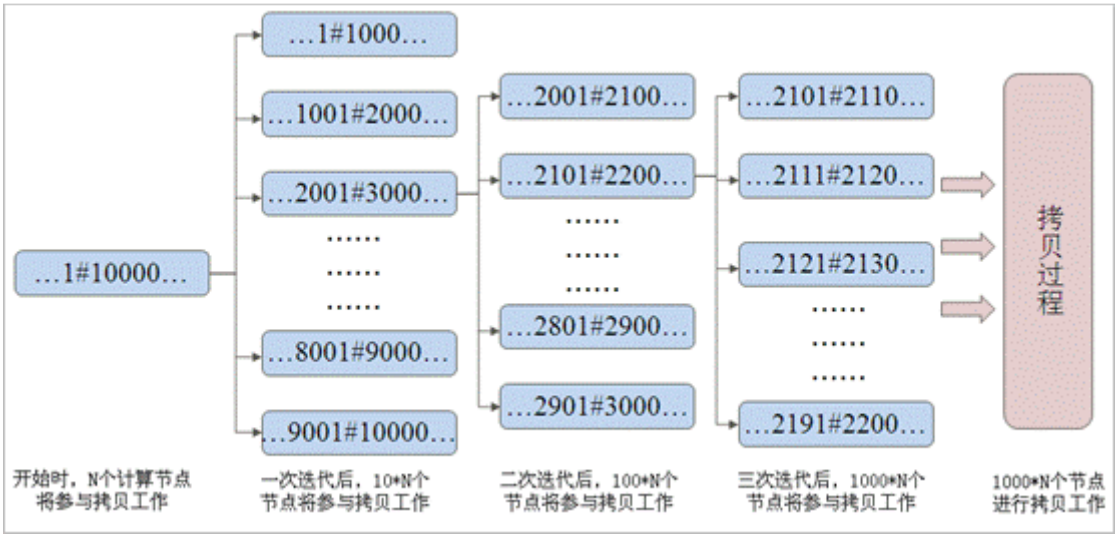

图6 Map迭代拷贝任务分发算法

为了解决上述问题，本发明提出了“Map迭代拷贝任务分发算法”来对每条记录（每个分段）的拷贝任务进行分发，这样有效的控制每个节点对每个分段的拷贝数量，同时更有效的使得更多的节点参与拷贝运算工作。

为了便于每条数据知道自己需要拷贝的分数，我们对公式（1）、（2）进行简单的修改：

$$k \quad L-i-1\#\lceil N_k/w \rceil - element\_list \quad i \in (1,2,\ldots,\lceil M_k/w \rceil) \qquad (3)$$

$$k \quad R-1\#\lceil M_k/w \rceil - j - element\_list \quad i \in (1,2,\ldots,\lceil N_k/w \rceil) \qquad (4)$$

式（3）中代表该记录（分段）需要拷贝$\lceil N_k/w \rceil - 1 + 1$份，拷贝标识号为1至$\lceil N_k/w \rceil$；同理解释式（4）。

这里，我们结合图7进行举例说明，假设...1#10000...是式（3）或式（4）的缩写形式，代表一条记录需要拷贝10000份，同时假设所有分段需拷贝的份数都是10000,那么初始时，将有N个节点参与拷贝工作。为了使得更多的计算节点参与拷贝工作，我们设计了此Map迭代拷贝任务分发算法。假设分发扩展率为10，则经过一次迭代后，文件大小扩大了约10倍，则大约有10*N个计算节点将参与拷贝工作，依次类推，三次迭代后，约有1000*N个计算节点参与拷贝工作，当有1000*N个节点参与拷贝工作时，每条记录被拷贝的最大份数为10，如图7所示。

l **迭代次数控制**

在现实大矩阵相乘中，由于大部分情况下矩阵都为稀疏矩阵，那么每行每列包含的元素个数就不一样，所以每个分段需拷贝的份数都不确定。这样我们就需要计算Map迭代过程的迭代次数，依次来控制Map迭代的过程。在此，我们利用图5所示存储在分布式缓存中的各个分段信息来得到最大分段数目，同时结合分发扩展率n，利用公式（5）来计算Map迭代的次数，依次来控制Map迭代过程。

$$N_{iteration} = \left\lceil \log_n Max\left(Max(I_1^L, I_2^L, \ldots, I_r^L), Max(I_1^R, I_2^R, \ldots, I_r^R)\right) \right\rceil - 1 \qquad (5)$$

最后计算模块

完成记录的拷贝工作后，我们还需要两轮Map-Reduce过程完成矩阵的运算。

l **第一轮Map-Reduce——分段拷贝与对应**

在此轮中，我们首先在Map阶段完成分发到的拷贝任务，若图（7）中的....2191#2200...格式符合式（3），则其原本的形态为：

$$k \quad R-2191\#2200-j-element\_list$$

在Map中执行拷贝工作后记录样式为：

k—i—2191　element_list

k—i—2192　element_list

……

k—i—2199　element_list

k—i—2200　element_list

若图（7）中的….2191#2200…格式符合式（3），则其原本的形态为：

$$k \quad R-2191\#2200-j-element\_list$$

在Map中执行拷贝工作后记录样式为：

k—2191–j　element_list

k—2192–j　element_list

……

k—2199–j　element_list

k—2200–j　element_list

经过此轮Map阶段，每个key(拷贝后每条记录的前半部分)对应两个Value，也就是L矩阵中的一个段与R矩阵中的一个段，同一个key的两个Value将在该轮的Reduce阶段进行汇合，汇合后如下所示：

$$k-i-j \quad element\_list\_(L,k,i)\#element\_list\_(R,k,j)$$

(L,k,i) 代表L矩阵第k列第i个分段，(R,k,j) 代表R矩阵第k行第j个分段，然后在下一轮Map–Reduce进行如图（4）所示的两个段的相乘工作。

l **第二轮Map–Reduce——相乘并汇总**

Map阶段对每条记录进行相乘运算，即将L中每个元素依次与R中每个元素相乘，若element_list_(L,k,i)中某个元素$L_{i,k}$与element_list_(R,k,j)中某个元素$R_{k,j}$相乘，则结果记录成“i–j value”格式。 然后每个Map结束后执行combine操作，combine操作与该轮Reduce操作一样，执行相同key的value相加，便得到了最终的矩阵运算结果。