哈尔滨工业大学

<<大数据分析>>

实验报告之大作业

问题二

分布式空间 Top-K 频繁关键字查询系统

(2019 年度春季学期)

姓名：	姜思琪、樊静(贡献度 1:1)
学号：	1160300814、1160300803
学院：	计算机学院
教师：	杨金华、王金宝

目录

一、问题描述

本文主要是通过建立索引，检索最频繁的关键字，根据查询数据信息和空间匹配程度,分布式进行提高效率，返回 Top-k 个最符合用户需求的结果。本文提出基于全局索引（KD 树）和本地索引（IR 树），实现分布式空间 Top-K 频繁算法。

给定空间文本对象数据集 D，对于对象 o 可以表示为一个三元组 o =< id，doc,loc >。同时给定一个对象查询 q。q 可以表示为一个元组 q=<locationArea，keywords>，已知基于空间范围的 Top-k 查询是指查询 q.keywords 为空的特殊情况，并从数据集 D 中返回满足 o.loc ∈q.locationArea，即满足查询的空间范围约束的关键字集合，并按照关键字词频高低降序排列。

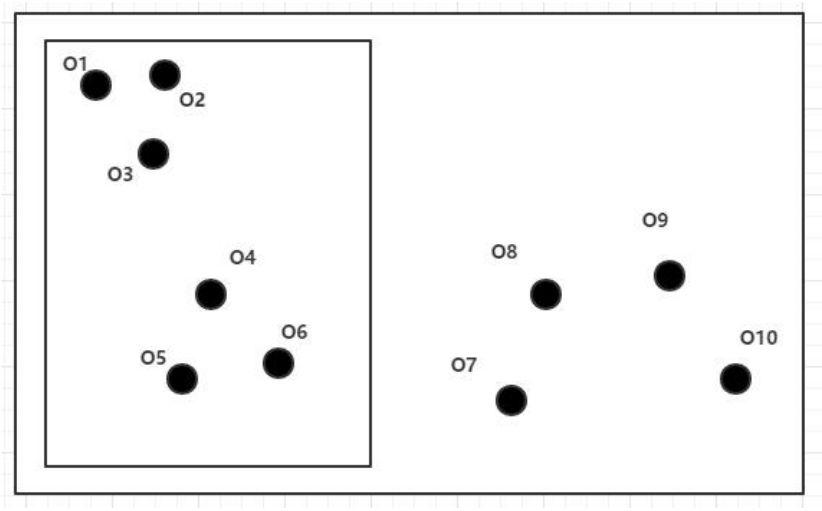

图 1.1 空间文本对象数据集及查询空间分布图

Ojbect	Terms	Object	terms
O_1	$\{t_1, t_5, t_8\}$	O_6	$\{t_4, t_6, t_9\}$
O_2	$\{t_2, t_3, t_6, t_8\}$	O_7	$\{t_1, t_2, t_3\}$
O_3	$\{t_3, t_7\}$	O_8	$\{t_5, t_6, t_7\}$
O_4	$\{t_1, t_4, t_8\}$	O_9	$\{t_1, t_7, t_9\}$
O_5	$\{t_1, t_3, t_6\}$	O_{10}	$\{t_2, t_6, t_8, t_9\}$

表 1.1 空间文本对象数据集词条分布

例 1:如图 1.1 所示，图中显示了空间文本对象数据集 D，q 的空间范围约束如图 1 中实线范围所示。表 1.1 中给出了数据集 D 中所有对象 o.doc 所包含的所

有词条（不显示词频，若词条重复则只显示一次）。则根据 D 和 q 可以得出最后结果如下表 1.2 所示。

Term	ObjectEntries	Freq	Term	ObjectEntries	Freq
t_8	$\{o_1, o_2, o_3, o_4\}$	10	t_4	$\{o_4, o_6\}$	4
t_9	$\{o_1, o_2, o_4, o_6\}$	8	t_2	$\{o_2\}$	3
t_6	$\{o_2, o_5, o_6\}$	6	t_5	$\{o_1\}$	3
t_1	$\{o_1, o_4, o_5\}$	5	t_7	$\{o_3\}$	2
t_3	$\{o_2, o_3, o_5\}$	5	t_{10}	$\{o_6\}$	1

表 1.2 空间文本对象查询结果

上述例子是对空间范围的 Top-k 查询的简单举例，实际场景则要复杂的多。主要体现在：

空间文本对象数据集 D 的规模问题。随着 D 数据规模的增加，传统的查询方式已经无法满足用户查询需求，需要寻找对数据伸缩性具有良好支持的索引结构来提高大规模对象的查询性能。当 D 的规模增加时，匹配的难度随之增加。

二、系统设计

2.1 存储

2.1.1 数据划分

是从内到外的整体分割，将数据以向量的方式存储，以坐标系的原点为起点，将整个坐标系由内向外划分为 m 个区域，将每个区域从外向内顺序编号为 S i,i＝1,2……，m，且第 1 区域的边界与坐标轴一起共同将所有数据对象都包括进去，对任意一个区域，该区域的每种属性的最大值相同，且每个区域的外围边界的坐标满足至少有一个轴的坐标值为该区域的属性的最大值，在设定第 1 区域的属性的最大值为 a_1 的前提下，则第 i 个区域的属性的最大值:

$a_i=a_1*((m-i+1)/m)^{1/d}$，i＝1,2……，m划分粒度不能过细。

除最外面一个区域，对其余每个区域，将属于该区域的且各个轴的属性均为最大值的点作为基点，将整个坐标系中的所有属性值都大于等于基点处的相应属性值的区域均划分出来，按照从外向内的顺序编号为 N i，i＝1,2，……，m－i，将上述新划分出来的区域作为判断区进行如下操作判断：

按照从小到大的编号顺序，依次判断 Ni>=k 是否成立，其中为 Ni 为 i 号判断区中的数据对象的个数，k 为算法返回的结果集中数据对象的个数；当 i 号判断区域满足上式成立，则结束判断，并将从外向内的 i 个区域作为搜索区域进行

搜索。

如果想要再进一步减少查询的数据量，可以对每个大区域再进行细分。

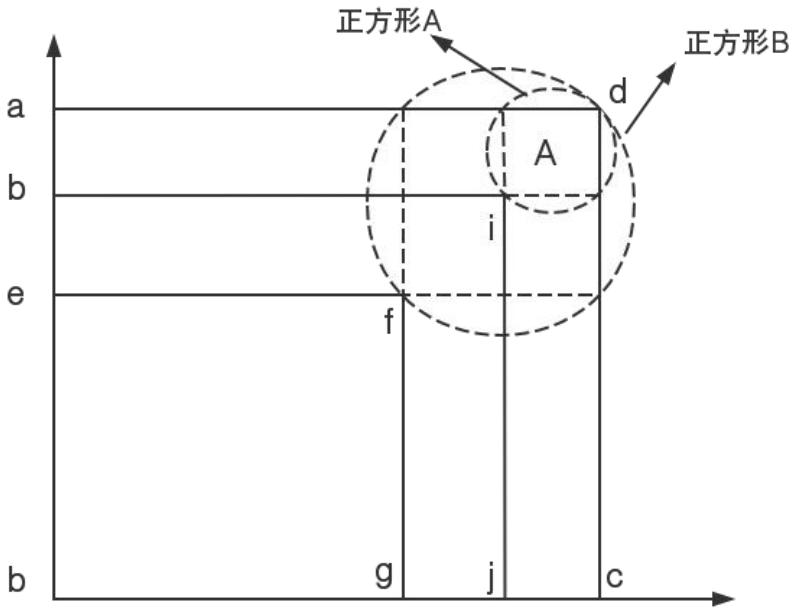

图 2． 1 数据粗粒度划分

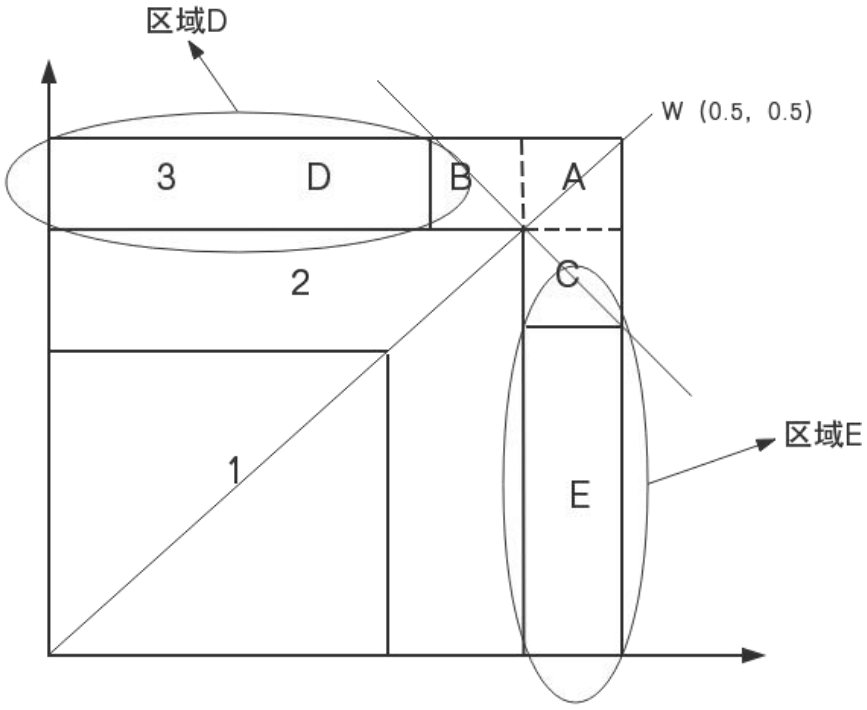

图 2.2 每个子空间细粒度划分

2.1.2 导入新的数据

2.1.3 数据备份

2.2 索引

我们采取的索引构建方法是双层索引构建：由全局索引和局部索引组成；上层的全局索引相当于索引系统的 Master 节点，它保存局部索引的元数据信息，能够根据用户提出的查询条件，找到可能包含查询结果的局部数据节点来响应用户请求；下层的局部索引保存具体的索引数据，相当于 Slave 节点，主要负责检索本地数据，为了避免额外的网络传输开销，局部索引保存在与其对应的数据节点上。

在双层索引中，局部索引主要负责对本地数据的组织和管理，每个数据节点构建一个局部索引，局部索引的基础上，创建了全局索引，由局部索引的部分数据信息组成，主要描述数据的数值范围和数据的存储位置。

索引的查找过程：当用户发出查询请求时，查询执行过程主要分为一下几步：首先，用户提出的查询请求发送到全局索引服务器，在全局 KD 树上进行检索，找到满足用户查询条件的 KD 节点，并由全局索引服务器将查询请求转发给相应的局部数据节点，如果查询失败，则直接给用户反馈失败消息。然后，接收到查询请求的局部数据节点开始并发的执行局部 R 树索引的检索。最后，检索得到的数据经过整合后返回给客户端，查询执行结束。

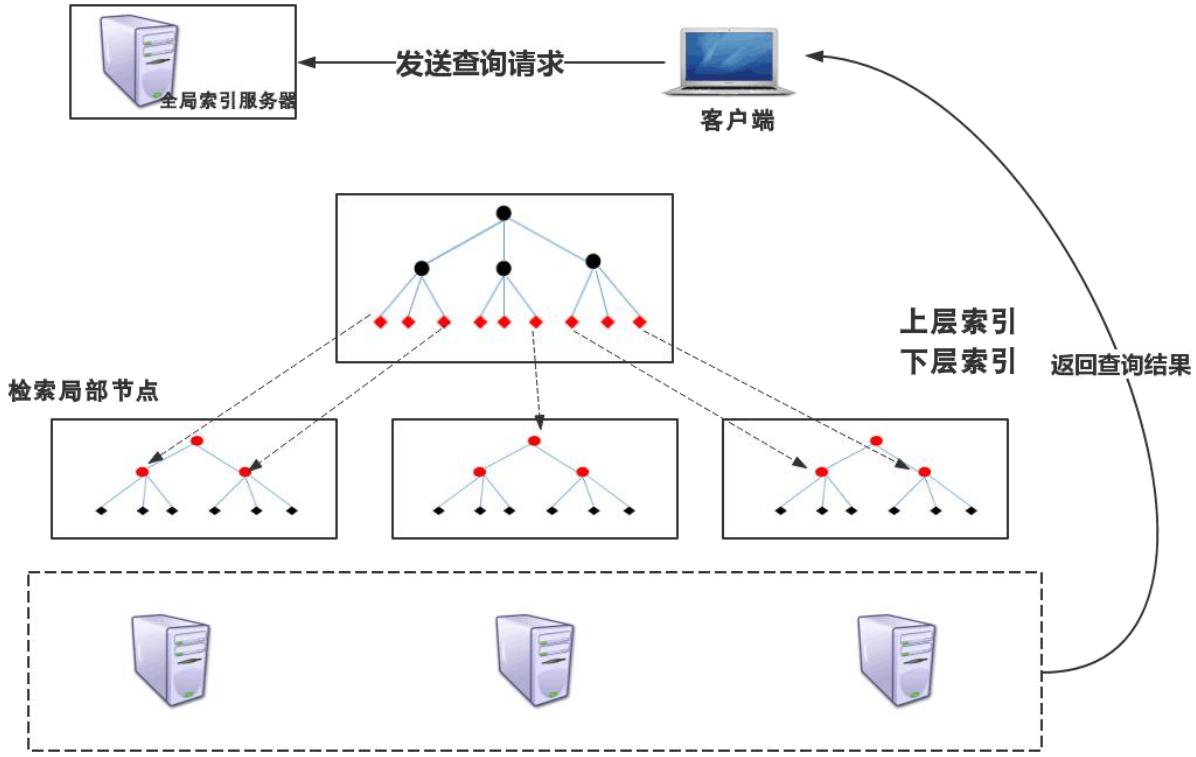

图 2.3 KD-IR 索引结构图

2.2.1 全局索引(可选)

原始的 KD 树是一棵平衡的二叉树，其本质是将 A 维的数据空间按维度分层递归的分割成多个小的子空间。树中的一个节点对应原始数据空间中的一个数据点，同时也代表一个空间分割点，它们将原始数据空间划分成相互独立，互不相交的子空间。而 R 树的非叶节点存储的是最小外包矩形，即节点的多维范围，KD 树则是通过多维数据点对多维空间进行划分，会出现划分节点与 R 树节点交叉的情况，可能会出现漏查的现象，所以在这里采用从一篇论文中看到的改良的 KD 树，改良的 KD 树采用了三叉树的实现方式，通过划分节点对各个维度上的范围进行限定，保证了查询的正确性。

我们的全局索引结构选择了 KD 树来实现，具体实现过程如下：

(1) 三叉树结构

从局部索引发布到全局索引层的 R 树节点是一个多维范围，包含每一个维度上的最大值和最小值。选取其中一个节点作为 KD 树划分节点，三叉 KD 树定义如下：

左子树：存储最大值小于划分节点数值的集合。

中子树：存储划分节点数值在最大值和最小值之间的集合。

右子树：存储最小值大于划分节点数值的集合。

(2) 节点类型

三叉 KD 树节点包括两种类型：划分节点和叶子节点。如图所示，划分节点存储维度和数值，叶子节点存储对应的 R 树节点。

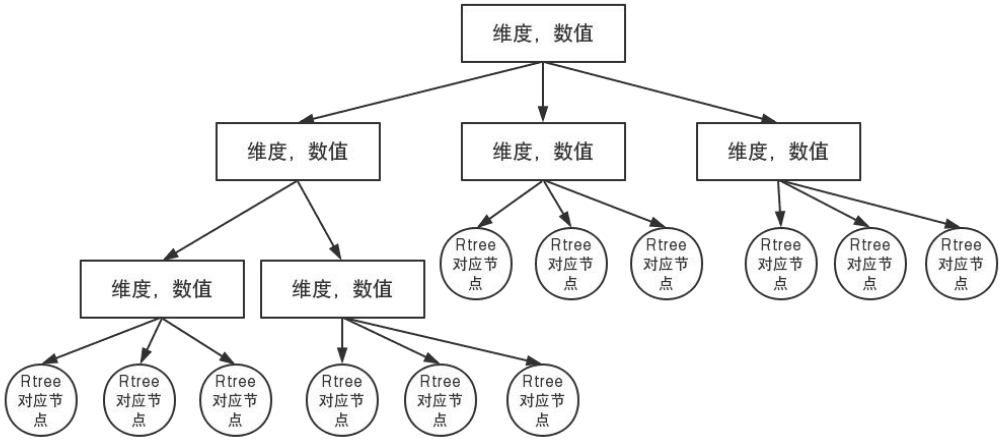

图 2.4 三叉 KD 树

(3)节点存储形式

KD 树的划分节点是一个五元组，定义为 KD-inner=（splitDim, splitValue, left, mid, right），其中 splitDim 表示进行空间划分的维度，即空间划分坐标：splitValue 表示当前的划分值,left mid right 表示当前节点的左子树，中子树和右子树指针。

KD 树的叶子节点存储具体的 IR 树节点的数据，是一个三元组，定义为 KD-leaf=（range,ip ,blk），其中，range 表示当前节点代表的多维数据范围：ip 表示存储该 R 树节点的数据服务器的地址；blk 表示存取该 R 树节点的磁盘块地址。

(4)构建算法：

算法 1 全局 KD 树构建算法

```
BuildTreeNode(rs,minsplitnode)
Input: rs -当前发布局部 R 树节点的数据服务器
Minsplitnode -全局索引节点分裂下限
1   /*Get published Rtree nodes */
2   For i=0 to rs.nodes
3   Put each Kdtree node into the queue
4   While the queue is not null
5      If length[Kdtree.curNode]<=minsplitnode
6         add queue.node[j] into Kdtree.curNode.leaf;
7      else split (Kdtree.curNode)
```

全局 KD 树索引的构建过程是一个递归过程，在算法 1 中，当待插入的索引

节点从队列中取出添加到 KD 树索引时，首先根据初始的空间划分元素将节点插入左子树、中子树或右子树，当节点个数超过了全局索引节点分裂的下限，则调用分裂函数 split()，KD 树节点进行分裂，直到队列中的所有节点插入完成为止

关于**节点分裂：**

应用最大方差法，选择具有最大方差的维度作为当前的划分维度，使得划分维度上的数据分布相对较广，且左右子树分布较为平均，中子树中数据量较少。在分裂过程中使用两个辅助数组（total 数组和 count 数组）分别存储多维空间中各个维度上的取值范围和每个维度值对应的多维数据点个数。节点分裂过程描述如下：

Step1: 扫描待插入索引节点，填充 total 数组和 count 数组。count 数沮对每个数值对应的节点个数进行统计。

Step2: 统计每个维度上划分的左右子树容量的差值。

Step3: 对中子树的节点数量与左、右子树之差进行加权计算，比较各维度的加权值，找到合适的划分维度和划分值，根据选定的划分维度和划分值对 K D 树节点进行分裂，将 R 树节点插入到 K D 树的相应子树中。

Step4: 当 KD 树节点包含的 IR 树节点较少，即小于节点分裂阈值时，停止分裂，完成节点队列中的所有节点的插入操作， K D 树构建完成。

查找过程举例：

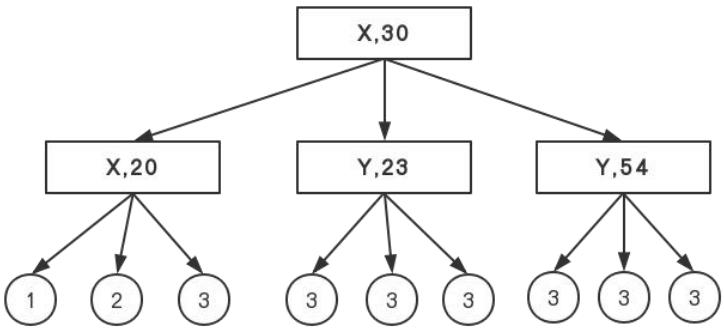

图 2.5 基于 KD 树的二维范围查找过程

假设用户想要查询的是 X[21:25]，Y[22:34］范围内的数据，如图 2.5 所示，在根结点中 X 维度的取值为 25<30。因此遍历途中左中子树。在最左子树中最小值 21 大于 20，则遍历中、右子树，找到 2、3 结点。在中子树中划分值为 Y ＝ 23，该划分值包含在 Y[22:34]区间内，需要遍历左、中、右子树，获得 4,5,6 节点。最终满足查询条件的节点为{2，3，4，5，6}。

2.2.2 局部索引

局部索引利用 IR 树实现。IR-Tree 结合了 R-Tree 和倒排文件这两种技术，可以有效地处理包含空间文本和文本双重属性的查询。IR-Tree 将这两者结合起来，以组合的方式生成混合索引结构。IR-Tree 本质上是一个 R-Tree，但每个节点都与一个倒排文件相关联。该倒排文件包含以该节点为根的子树中包含的所有对象。根据图 2.1 中 R-Tree 示例的相关数据，生成对应的 IR-Tree 示例如图 2.2 所示：

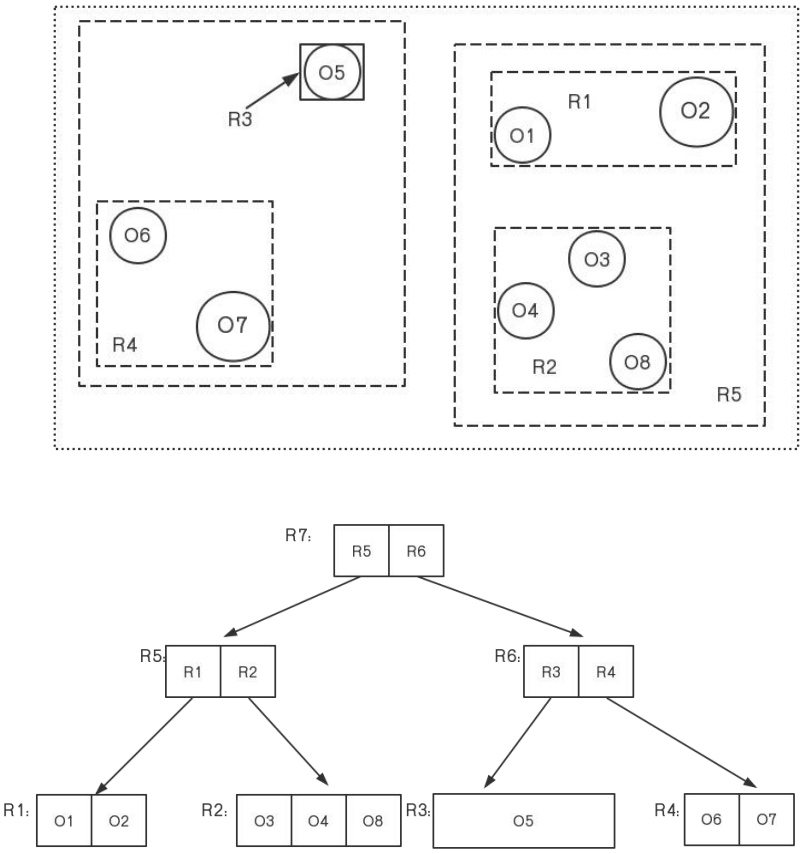

图 2.6 空间索引 R-Tree

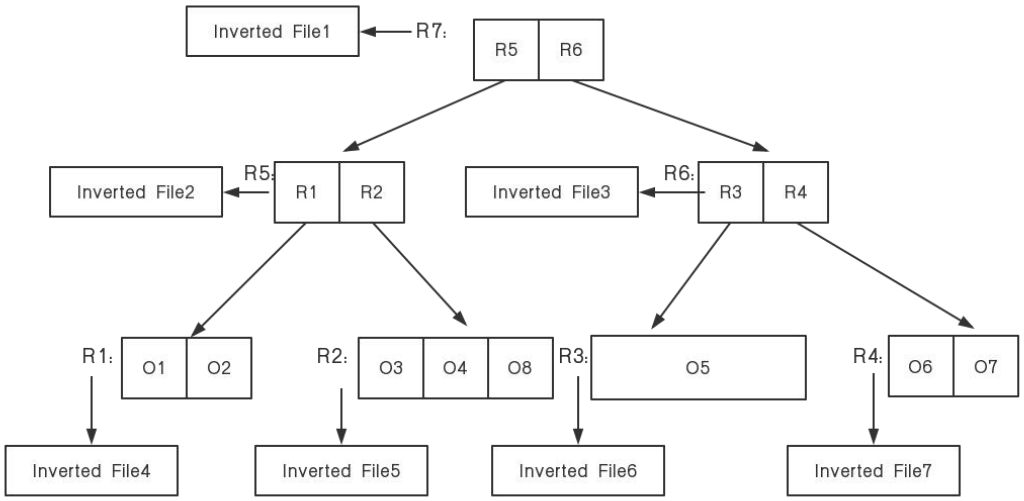

图 2.7 空间文本索引 IR-Tree

在 IR-Tree 中，叶节点 N 包含一个三元组(O, rectangle,O.id)，其中 O 表示空间文本对象；rectangle 表示对象 O 的边界矩阵，即 MBR；是 O.id 对象 O 的文档标识符，能唯一标示对象。另外，叶节点还包含一个指向索引对象集的倒排文件的指针。图 2.2 中 IR-Tree 示例对应的倒排文件如下表 2.2 和表 2.3 所示。

倒排文件由两个主要部分组成：第一部分是节点空间中文本对象集合中所有词汇的列表 T={t0,t1,...,tn-1}，其中 n 是单词的数量。第二部分是一个二元组<d,wd,t>,d 表示存在词汇 t 的空间文本对象对应的关键字集合,wd,t 表示出现在集合 J 中词汇/对应的权重。

Vocabulary	Inverted File 1	Inverted File 2	Inverted File 3
Chinese	<R_5.doc,7>,<R_6.doc,4>	<R_1.doc,5>,<R_2.doc,7>	<R_3.doc,4>,<R_4.doc,1>
Spanish	<R_5.doc,5>,<R_6.doc,4>	<R_1.doc,5>,<R_6.doc,3>	<R_4.doc,4>
restaurant	<R_5.doc,7>,<R_6.doc,4>	<R_1.doc,5>,<R_6.doc,7>	<R_3.doc,4>,<R_4.doc,4>
food	<R_5.doc,1>,<R_6.doc,1>	<R_2.doc,1>	<R_4.doc,1>

表 2.1 Posting lists for Inverted File 1,2.3

除此之外，IR-Tree 的非叶子节点 R 同样关联对应的倒排文件，也可以用一个三元组表示(cp，rectangle, cp.id)，其中 cp 是 R 节点对应的子节点的地址，rectangle 是了节点中所有矩形的最小边界矩形(MBR），cp.id 是对应的标识符。

Vocabulary	Inverted File 4	Inverted File 5	Inverted File 6	Inverted File 6
Chinese	<O_1.doc,5>	<O_3.doc,7>	<O_5.doc,4>	<O_7.doc,1>
Spanish	<O_2.doc,5>	<O_8.doc,5>	null	<O_6.doc,4> <O_7.doc,1>
restaurant	<O_1.doc,5>	<O_4.doc,7>	<O_7.doc,4>	null

	<O_2.doc,5>	<O_5.doc,4> <O_8.doc,3>	<O_6.doc,3>	
food	null	<O_3.doc,1> <O_4.doc,1>	null	<O_7.doc,1>

表 2.2 Posting lists for Inverted File 4,5,6,7

2.3 算法

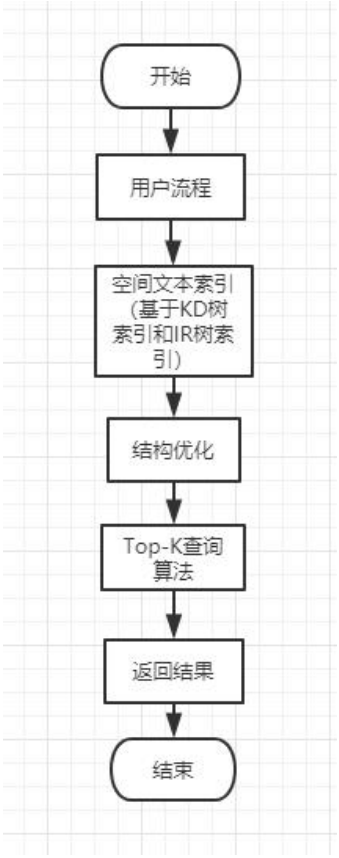

图 2.8 空间 Top-K 查询优化算法流程图

根据数据集是否支持随机读取，将算法主要分为两类：随机访问 (Random Access, RA)和顺序访问(Non Random Access，NRA)。

2.3.1 随机访问 Top-k 算法：RA(Random Access)

已知有序列表集 $L=\{l_0,l_1,.....,l_{n-1}\}$，其中 n 为列表集中列表的个数，且 li = { ($term_0$,$score_0$），($term_1$,$score_1$），.....，($term_0$,$score_0$）}，其中 term 表示列表中需要进行 Top-k 排序的条目，score 为该条目的得分，列表集 L 中每一个列表均按照 term 的 score 大小进行排序。

定义 2-2: 随机访问(Random Access)。算法迭代访问列表 l 中的第 i 项 term，

并随机访问列表集 L 中所有列表中相同的 term,并计算该 term 的总得分作为本次迭代的阈值 0。访问到下一个 term 时，若 term 的总得分高于阈值θ，则更新阈值θ，否则抛弃该 term。当找到 k 项总得分高于或等于阈值θ的时算法停止。
根据表 2.1 和 2.2 中定义，可设计一个 RA 算法示例如下表 2.3 所示：

O_1	O_2	O_3	Sum	Total Sum
(Chinese,0.9)	(Spanish,0.9)	(Spanish,0.9)	Sum(Chinese)=1.9	TS = 2.7
(restaurant,0.8)	(food,0.8)	(Chinese,0.9)	Sum(Spanish)=2.2	TS = 2.5
(food,0.6)	(restaurant,0.7)	(eat,0.8)	Sum(restaurant)=1.8	TS = 2.1
(Spanish,0.4)	(eat,0.5)	(food,0.5)	Sum(food)=1.9	
(eat,0.1)	(Chinese,0.1)	(restaurant,0.3)	Sum(eat)=1.4	

表 2.3 RA 算法示例

如上表 2.3 所示，算法过程如下：
(1) 算法开始先从第一行顺序扫描，计算出这行总的值大小 TS=2.7,并随机读取到所有列中 Chinese 和 Spanish 的这两个 term 的得分，计算并保存，由于最大值 Sum(Spanish)=2.2<TS=2.7,因此算法继续；
(2) 第二行算出 TS 大小 2.5,并随机读取计算 restaunt 和 food 的得分，计算并保存，最大值 Sum(Spanish)=2.2<TS=2.5,因此算法继续；
(3) 第三行算出 TS 大小 2.1，随机读取计算出 eat 的得分，最大值仍是 Sum(Spanish)=2.2>TS=2.1，找到最大数值 Sum(Spanish)。
(4) 重复上述 1-3 步，直到找到 k 项最大值 Sum 大于阈值 TS 时停止。

2.3.2 空间范围的 Top_k 查询（STRA）

根据时空文本对象的时空范围进行 约束，检索满足空间约束条件的关键字或对象。查询对象的空间范围是约束条件，必须 精确匹配，再根据一定的排序规则（关键字词频或 TF-IDF）返回符合条件的数据。

采用文本关键字的热度和重要性来评价指定范围内文本对象的得分，主要的评价指标是文本对象中文本关键字的 TF-IDF，常用于关键字重要性的计算。计算公式如下：

$$Tfidf_{i,j} = tf_{i,j} * idf_i$$

其中，$tf_{i,j}$ 是指文木关键字 t_i 在文件 d_i 中出现的词频，idf_i 是指文木关键字 t_i 逆向文本频率。计算公式如下：

$$tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}, idf_i = log\frac{|D|}{|j:t_i \in d_j|}$$

其中，$n_{i,j}$是指文本关键字 t_i 在文件 d_i 中出现的次数，分母 d_j 是文件中所有文本关键字出现的词数之和；|D|是语料库中文件总数，$|\{j:t_i\}|$是包含 t_i 的文件数目。最后 可以得出指定空间文本对象 O_i 的总得分，计算公式如下：

$$o_i.weight = \sum_{0 \le j < n}^{i} tfidf_{j,i}$$

基于时空范围的 Top-k 查询算法 STRA 的算法步骤如下：

(1) 初始化所有数据结构，包括结果队列、IR-Tree 节点队列。

(2) 将当前节点和查询条件放入 IR-Tree 节点优先队列，建立 IR-Tree。并根据定义计算频繁时空文本对象数据集 Dfreq* ;

(3) 若 D*freq 非空则优先访问，若为空则遍历 IR-Tree,找到满足约束条件且得分 g 最高的 k 个点。若结果优先队列中节点个数不满足 k，则根据公式 3-5 计算该节点的 得分，并将节点加入结果队列。并继续遍历 STR-Tree:

(4) 当结果优先队列中节点个数大于等于 k，并且当前访问节点 e 的得分小于结果队列 中第 k 个点的得分 Min，算法终止。这样能保证 STR-Tree 中的剩余节点中，不会 再含有比当前第 k 个点得分更大的节点。算法伪代码如下算法所示：

算法 2 基于空间范围的 Top-K 查询算法 STRA(q, IR-Tree)

Input：用户查询 q, 空间文本索引 IR-Tree 的根节点 RootNode

Output：k 个空间文本对象

```
1：Result <- NewPriorityQueue();       //初始化结果队列
2：D*    <- NewPriorityQueue();       //初始化 IR-Tree 节点队列
3：D*.Enqueue(q.locationArea,RootNode,0);
4：D*freq <- Calculation(D*);       //根据定义计算 D*freq
5：while Result.size<k || e.key> min do
6：   e <- D*freq.isEmpty()?D*.Dequeue():D*freq.Dequeue() ;
7：   if e is internal node then
8：         for each node n ∈e do
9：               D*freq.isEmpty()?D*.Eequeue(n,n.weight):D*freq .Enqueue(n,n.weight);
10：                    Update(min);
11：         end for
12：         else if
```

```
13：            for each e* in e do
14：                if ORankByWeight(e*,k) >min then
15：                    Result.Enqueue(n,n.weight);      //根据公式计算得分并插入结果集
16：                    D* freq.isEmpty()?D*.Eequeue(e*,e*.weight):D* freq .Enqueue(e*,e*.weight);
17：                end if
18：            end for
19：        end if
20：return Top-k objects in Result;
```

三、系统工作流程

图 3.1 系统工作流程

将数据读入并划分，划分到不同的结点上利用分布式提高效率。然后开始构建索引。建立 R-Tree，每个节点都与一个倒排文件相关联，该倒排文件包含以该节点为根的子树中包含的所有对象，从而构建 IR-Tree 实现局部索引。局部索引主要负责对本地数据的组织和管理，每个数据节点构建一个局部索引，局部索引的基础上，利用 KD 树结构创建了全局索引，由局部索引的部分数据信息组成，主要描述数据的数值范围和数据的存储位置。

查询算法是从全局 KD 树的根节点出发，先将根节点入栈，KD 树的每个节点包含划分维度和划分值，若在该维度上，要查找的多维关键字的值小于划分值，则左、中子树入找；若大于划分值，则右、中子树入栈。依次按层次向下遍历，直至找到所有满足查询条件的 R 树局部索引节点为止。

首先通过全局索引层的搜索找到可能包含查询结果的 R 树索引节点，根据索

引节点中保存的ＩＰ地址找到对应的局部数据节点。然后在相应的局部数据节点上执行局部Ｒ树索引的搜索，找到满足查询条件的所有数据.