

边不相交划分的分 布式RDF系统

Edge-disjoint Partitioning-based Distributed
RDF Graph Management



Peng Peng

Hunan University, China

hnu16pp@hnu.edu.cn



湖南大学
HUNAN UNIVERSITY



目录

01

背景

02

独立于查询日志的方法

03

基于查询日志
的方法

04

结论





湖南大学
HUNAN UNIVERSITY

第一部分

背景



资源描述框架

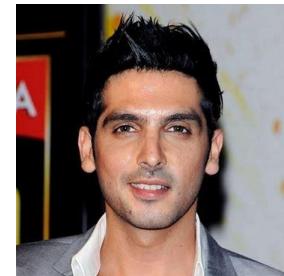
这是一个被广泛用于知识图谱的数据模型

一切事物都表示成一个具有唯一名称的资源

可以定义资源的属性

可以定义与其他资源的关系

dbpedia:Zayed_Khan



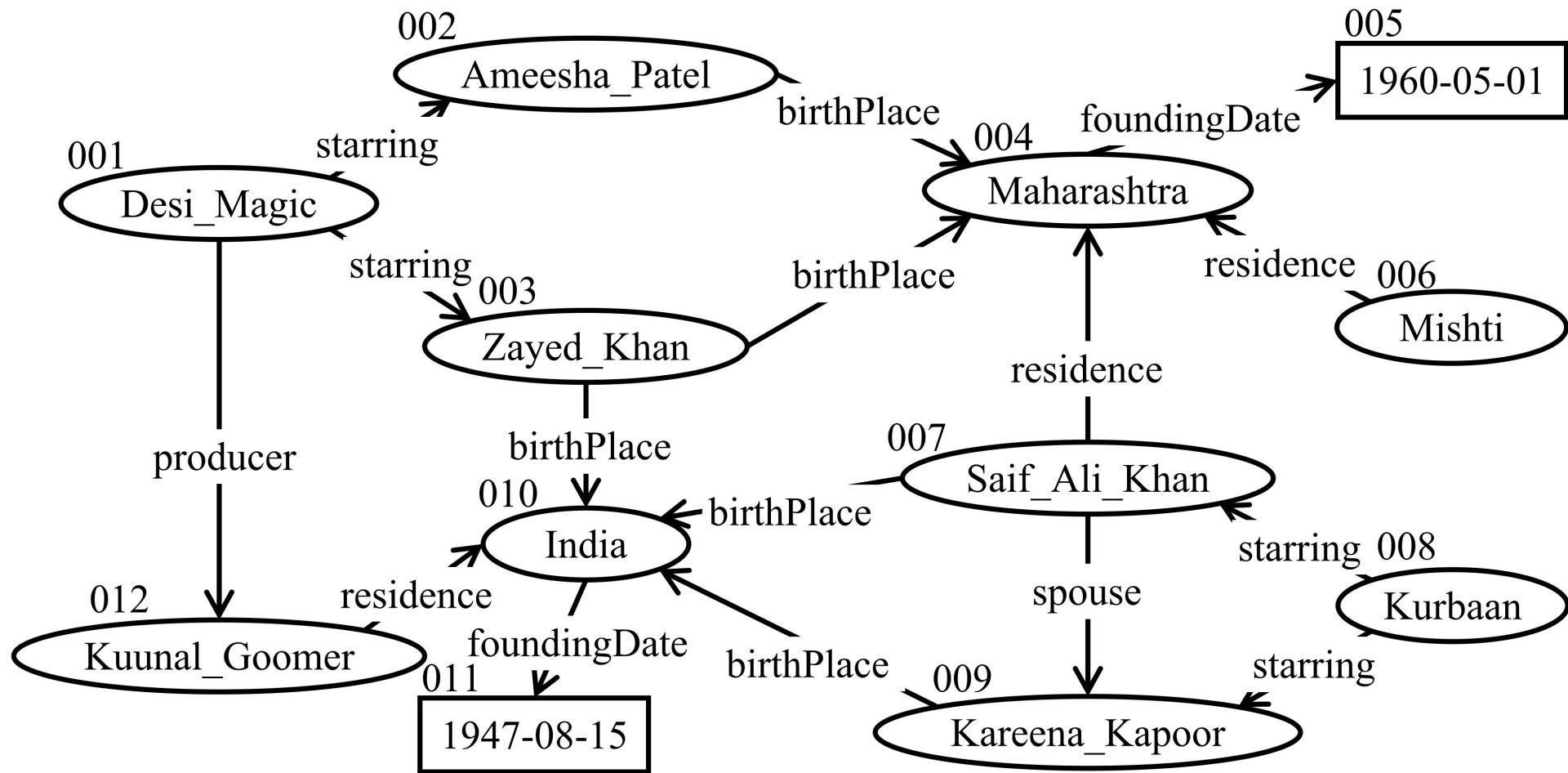
dbpedia:name "Zayed Khan"@en
dbpedia:dateOfBirth "1980-07-05"

dbpedia:birthPlace



dbpedia:Maharashtra

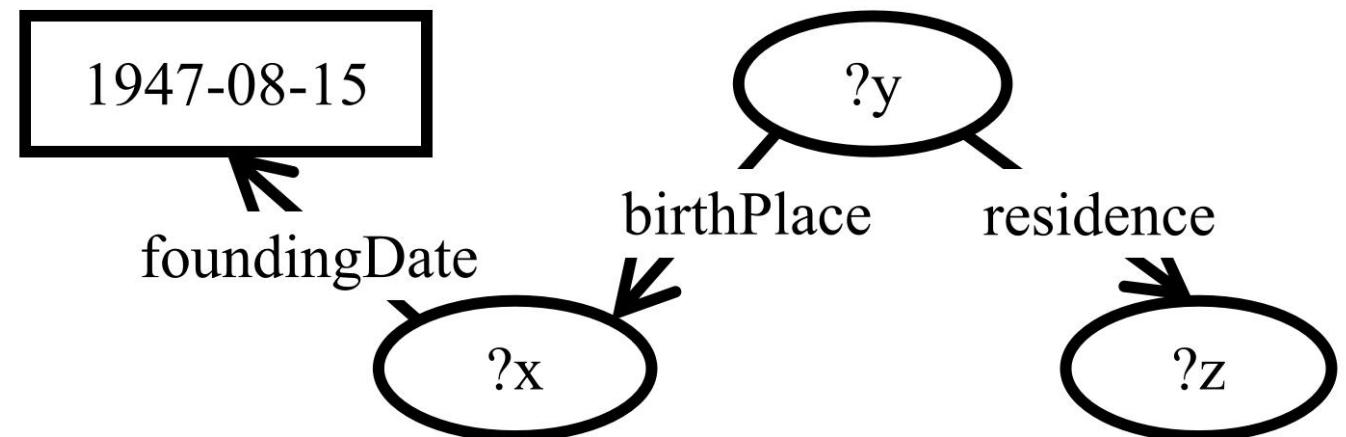
一个RDF数据集可以表示为一个图



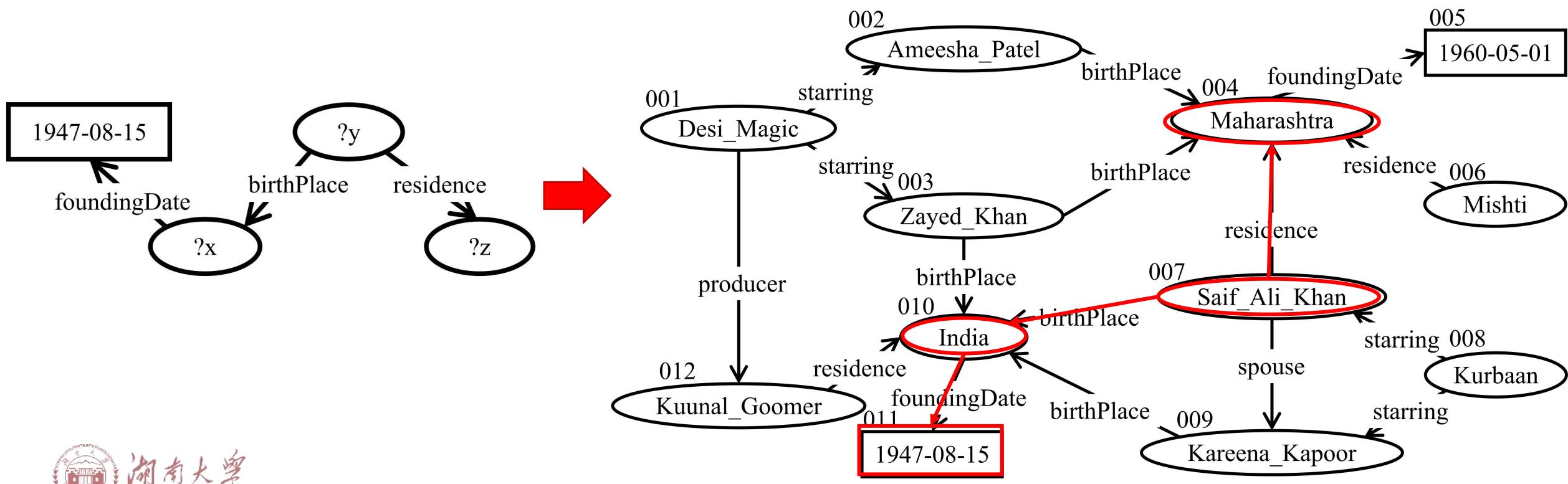
查询模型 - SPARQL协议和RDF查询语言

SPARQL查询是一组带有变量的三元组模式

```
Select ?x where {  
    ?y      residence      ?z .  
    ?y      birthPlace     ?x .  
    ?x      foundingDate   1947-08-15 . }
```



回答SPARQL查询等同于使用同态子图匹配来进行子图匹配



当前，RDF数据集变得越来越大



Max-Planck-Institute

284 million triples



Metaweb Company
acquired by Google in 2010

2.4 billion triples



Leipzig University
University of Mannheim
OpenLink Software

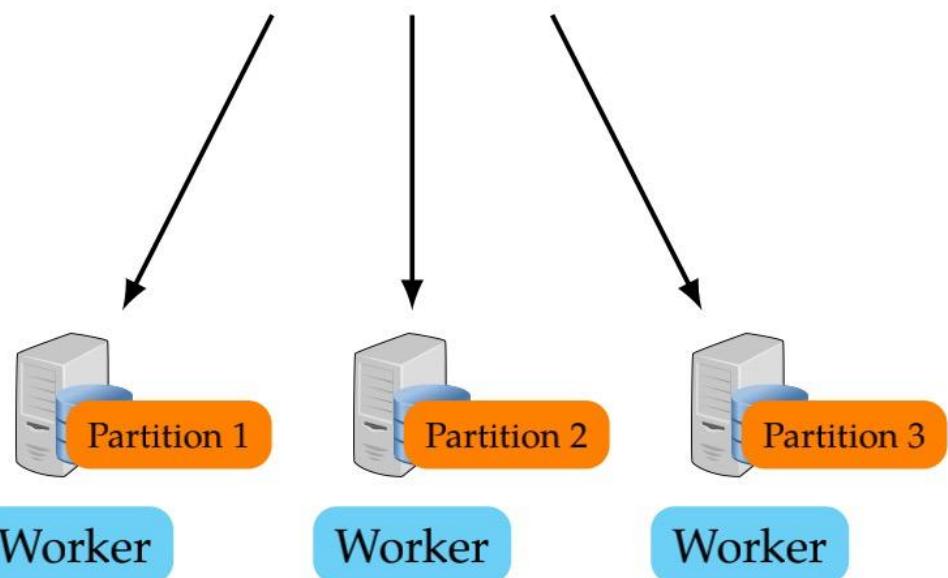
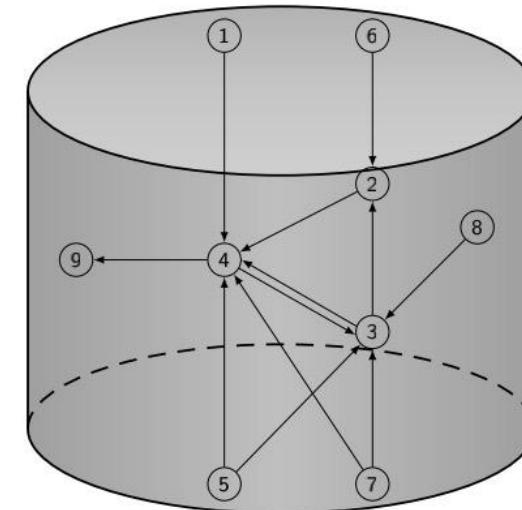
9.5 billion triples

设计一个分布式RDF系统来管理大型RDF数据集是非常重要的！

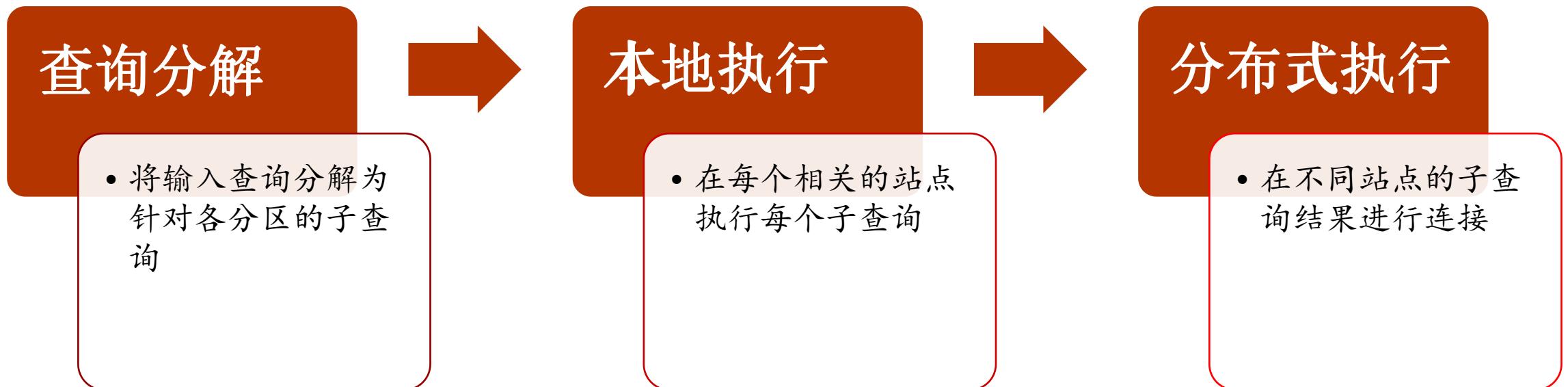
基于分区的架构

一个RDF图通过不同的站点进行分区

目标：尽可能减少站点间通信，实现并行化查询处理。



以尽量减少分区间连接的方式对数据和查询进行分区

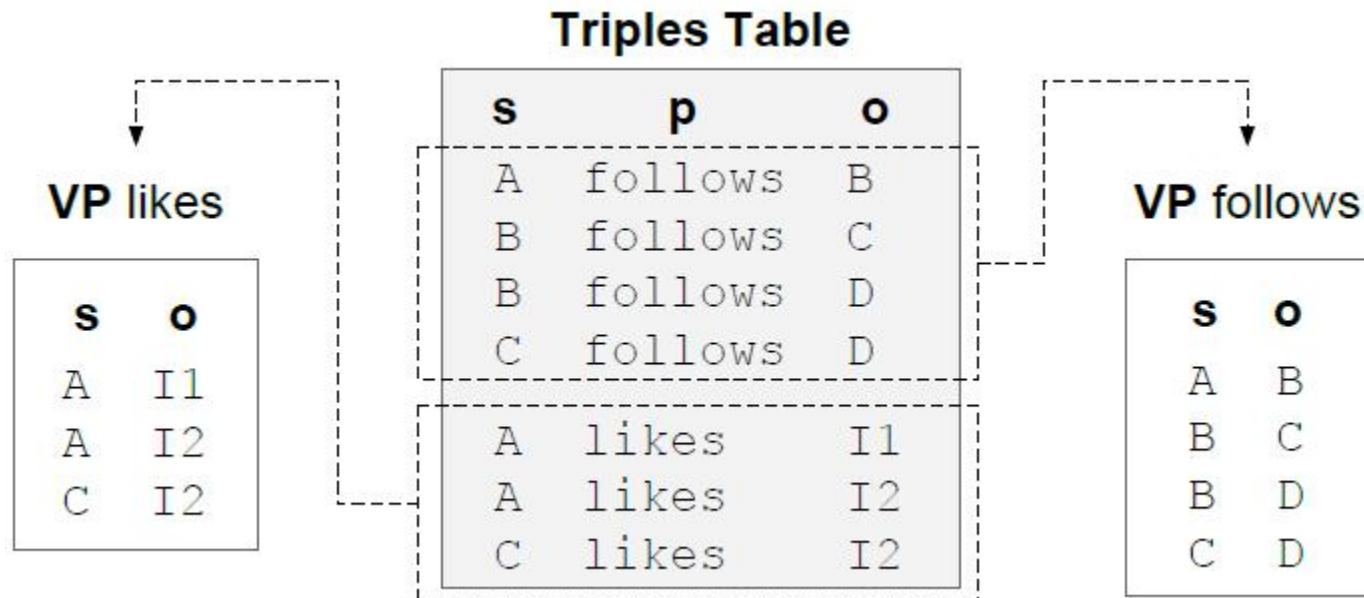


RDF图划分方法可以分为三种类型：

- 点不相交：将每个顶点放入一个分区。现有方法的目的是最小化边切割，而不是最小化分区间连接
- 边不相交：将每条边放入一个分区。现有方法在许多基于云的系统中广泛使用，以剪枝不相关的分区并避免在云中进行过多的扫描，但并不专注于避免分区间连接
- 其他：考虑额外信息，如查询日志

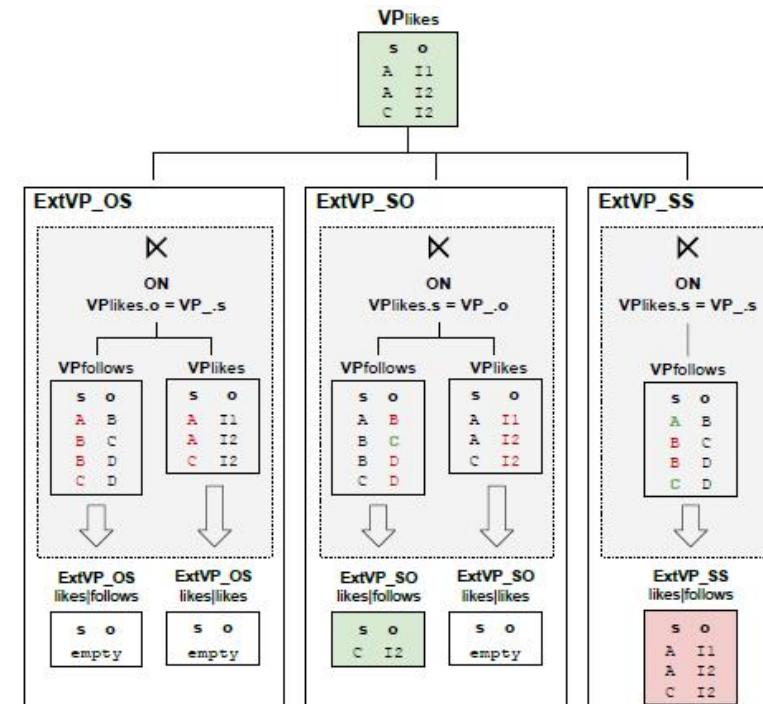
S2RDF

- S2RDF利用Spark的关系数据库接口进行知识图谱数据管理
- S2RDF利用垂直划分对知识图谱数据进行划分



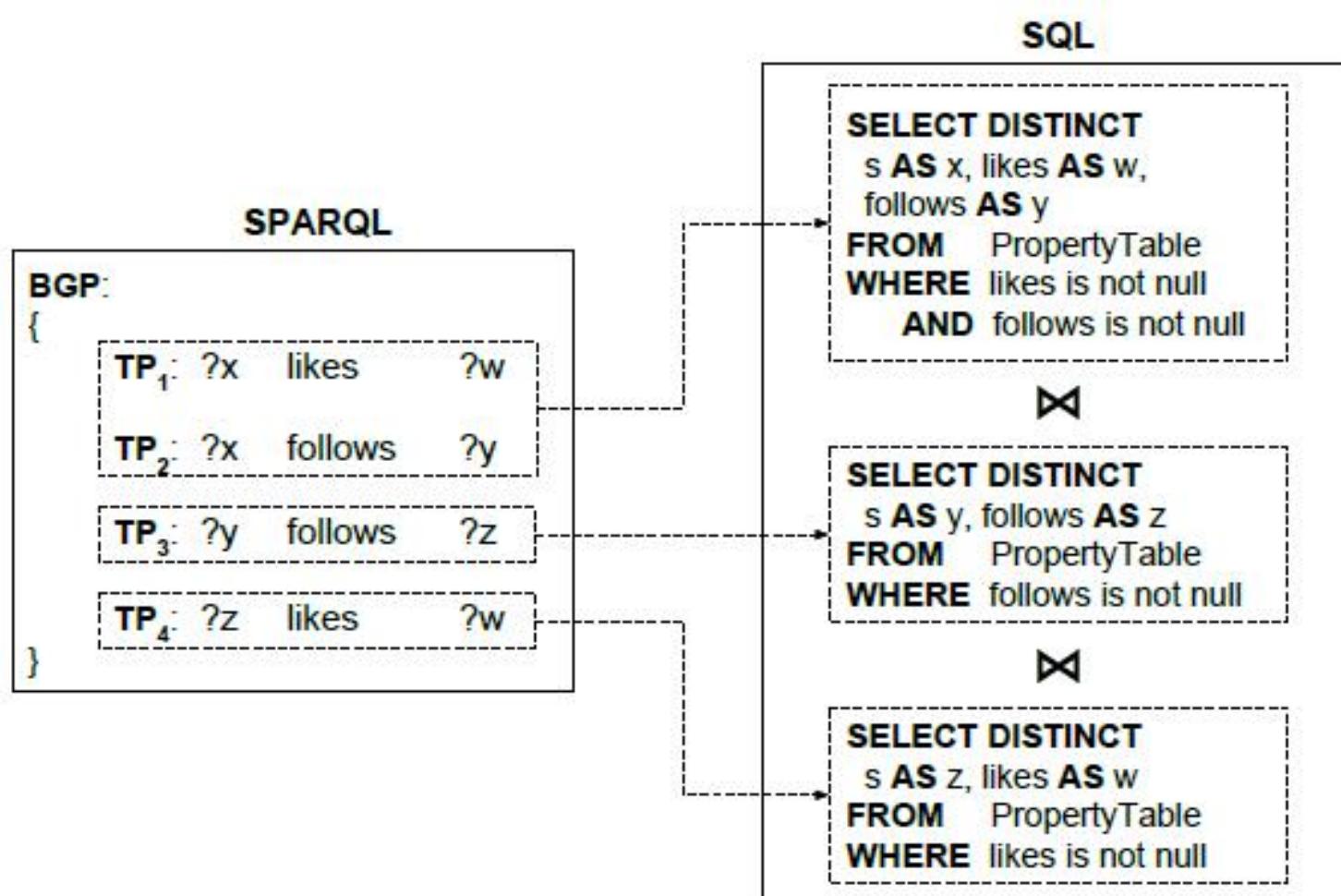
S2RDF

- 在基本垂直划分基础上，S2RDF物化了部分垂直划分数据表之间的连接结果并也存储在关系数据表



S2RDF

- 查询分解成基于垂直划分的子查询，并转化成SQL





第二部分

独立于查询日

志的方法



本工作提出了MaxLocJoin-EDP来优化复杂查询处理

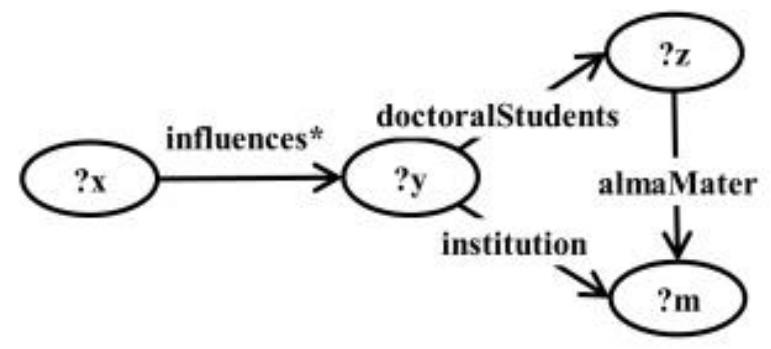
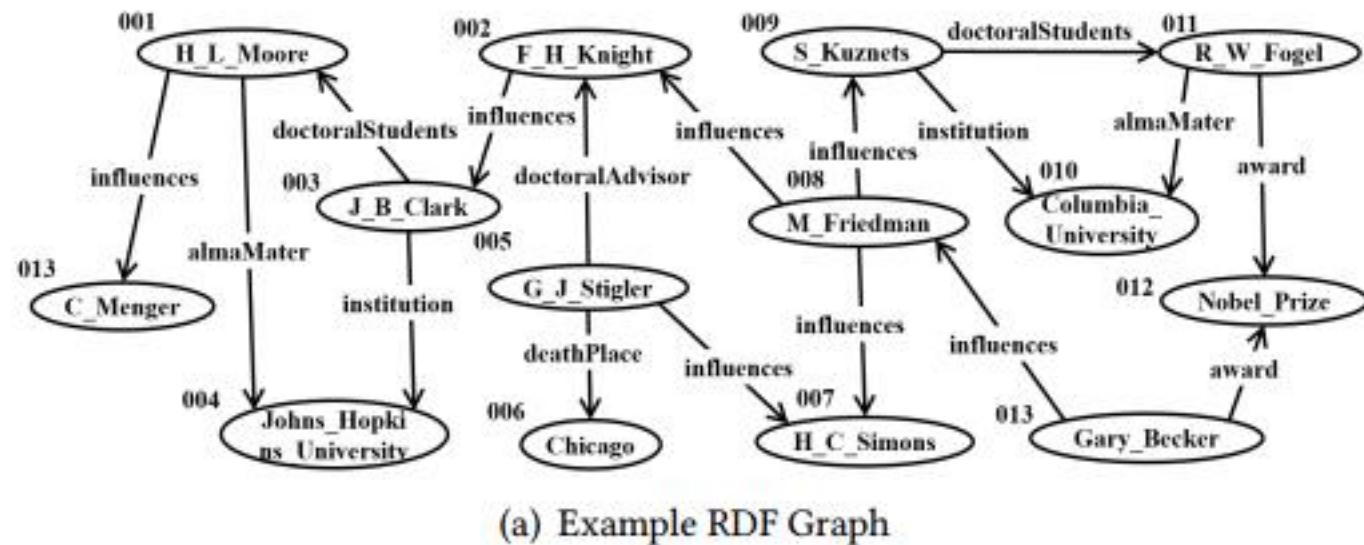
目标：保证单属性递归约束不跨分片的基础上，尽量让更多查询可独立执行

SPARQL查询中的基本单元是三元组模式，SPARQL查询通常包含一组三元组模式。每个三元组模式也是一个三元组，允许在主题、属性或对象位置中使用变量

SPARQL查询也可以看作是带有变量的查询图，回答SPARQL查询等同于在RDF图中查找查询图的子图匹配。

我们将一个属性后面的星号(*)或加号(+)表示为单属性递归约束。形式上，这个约束对应于性质的Kleene星号或加号闭包。包含具有这种单属性递归约束的三元组模式的查询称为复杂查询

图1(a)显示了一个从Wikidata提取的示例RDF图，图1(b)显示了一个具有单属性递归约束的三元组模式复杂查询，其中三元组模式中的属性位置是属性influences后跟一个星号(*)。匹配这个三元组模式是为了在RDF图中找到所有边都具有属性influences的路径。



(b) Example Complex Query

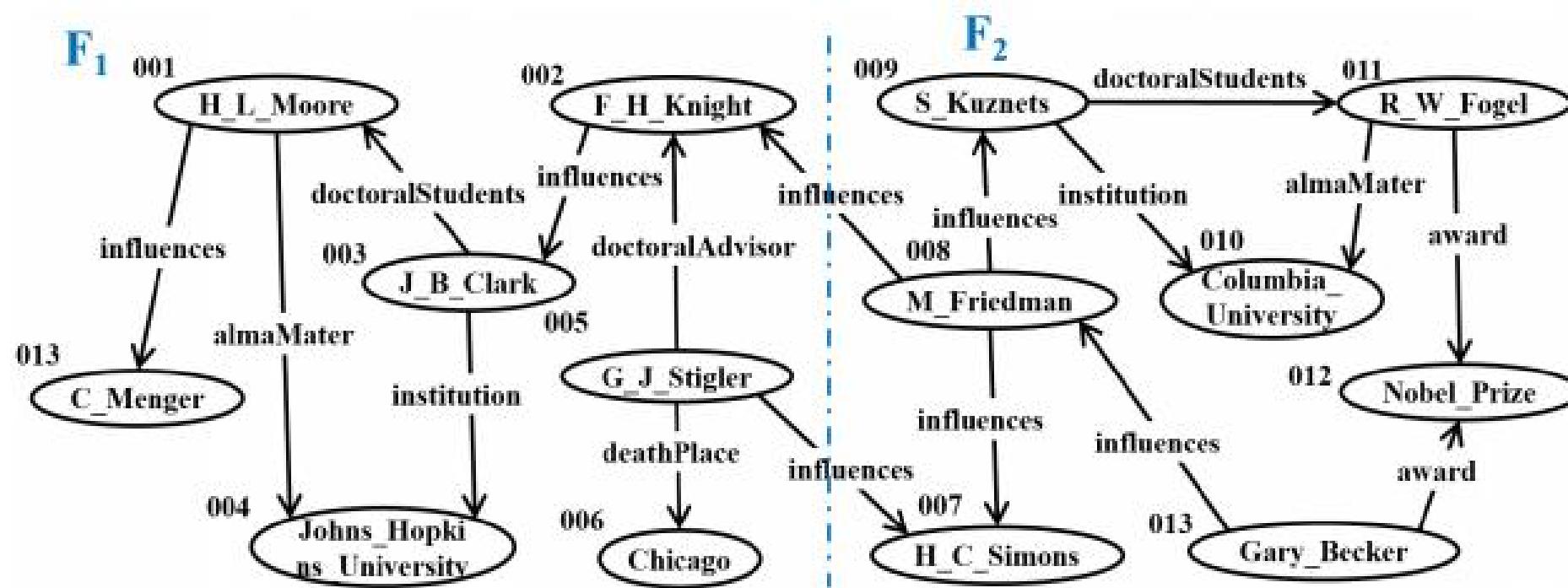
从Wikidata数据集公开查询日志中，我们发现相当大比例的查询(范围从10%到50%)符合复杂查询的条件

Date	Percentage of Complex Queries
2017/06/12 – 2017/07/09	14.13%
2017/07/10 – 2017/08/06	13.46%
2017/08/07 – 2017/09/03	11.15%
2017/12/03 – 2017/12/30	50.59%
2018/01/29 – 2018/02/25	31.15%
2018/02/26 – 2018/03/25	29.09%

在典型的分布式RDF系统中，RDF图G被分成一组称为分区的子图，表示为 $\{F_1, F_2, \dots, F_m\}$

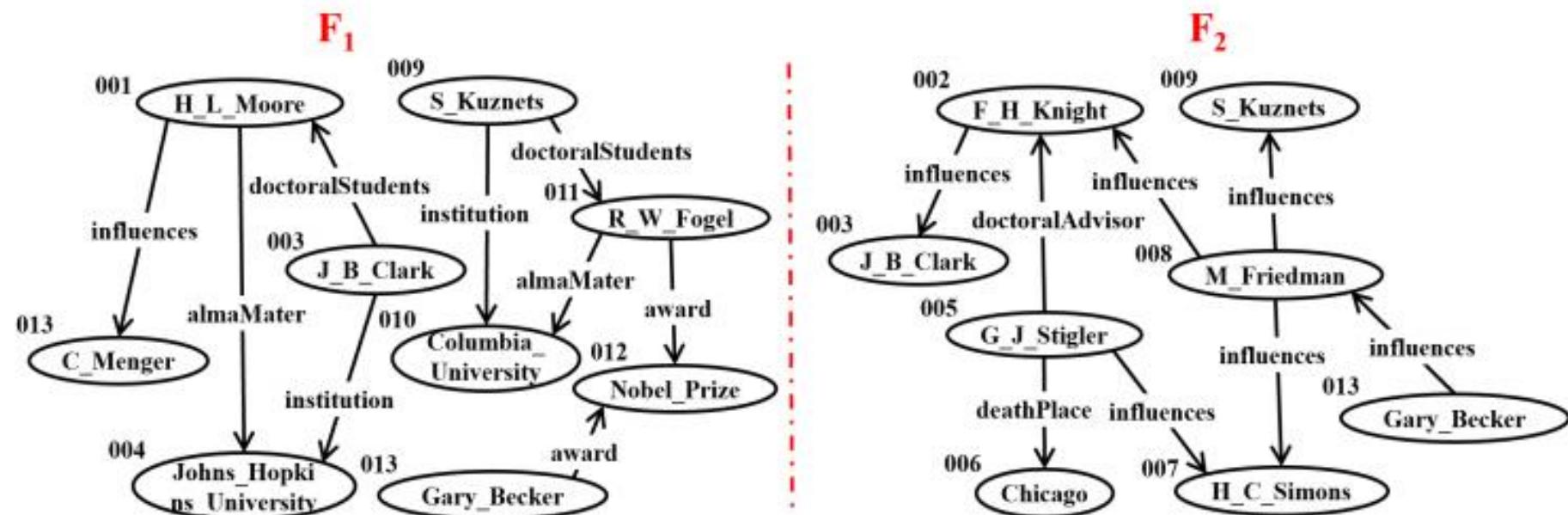
对于查询，如果匹配完全包含在单个分区中，我们将其称为内部匹配，而交叉匹配则跨越多个分区。

图2(a)显示了由Metis生成的一个顶点不相交分区，其目的是最小化两个分区之间的边数



(a) Example Minimum Edge-Cut Vertex-Disjoint Partitioning (Metis)

图2(b)为本文提出的方法生成的边不相交分区，其目的是减少分布式复杂查询处理中分区间连接的数量。



(b) Example Our Edge-Disjoint Partitioning (MaxLocJoin-EDP)

在现有的边不相交分区方法中，获得最终结果需要进行分区间连接，而且成本较高

我们引入了一种新的边不相交分区方案，该方案专门用于减少分布式SPARQL查询处理中的分区间连接数量，特别关注复杂查询

边不相交划分方案(MaxLocJoin-EDP): 在保证由具有特定性质的边所引起的子图的每个弱连通分量的同时，最大限度地增加了本地可连接属性的数目

在RDF图的边不相交分区中，如果所有具有 L_{local} 中属性的边引起的子图的每个弱连接分量都在一个分区中，则我们将一组属性 L_{local} 称为**本地可连接属性**（Locally Joinable Property）

可以证明，这种分区方法可以有效地避免复杂查询的两部分，即单属性递归约束和剩余的没有单属性递归约束的子查询的分区间连接

定义2.1(RDF图)

RDF图表示为 $G = \{V, E, L, f\}$, 其中, V 是一组顶点, 对应于RDF数据中的所有主体和客体; $E \subseteq V \times V$ 是对应于RDF数据中所有三元组的有向边的多集; L 是一组边缘标签; $f : E \rightarrow L$ 是一个标签映射, 其中对于每条边 $e \in E$, 其边标号 $f(e)$ 是其对应的属性。

定义2.2(路径)

两个顶点 $u_1, u_l \in V$, 我们定义一个序列 $\pi = (u_1, u_2, \dots, u_{l-1}, u_l)$ 来表示一个路径 $u_1, u_2, \dots, u_{l-1}, u_l$ 与 v_l , $\{u_1, u_2, \dots, u_{l-1}, u_l\} \subseteq V$ 和 $\{\overrightarrow{u_1 u_2}, \overrightarrow{u_2 u_3}, \dots, \overrightarrow{u_{l-1} u_l}\} \subseteq E$

边不相交分区将每条边分配给一个分区(分区是边不相交的)。在形式上，我们定义RDF图的分区如下，其中 u 和 $\overrightarrow{uu'}$ 表示顶点和边， k 是图分区的数量。

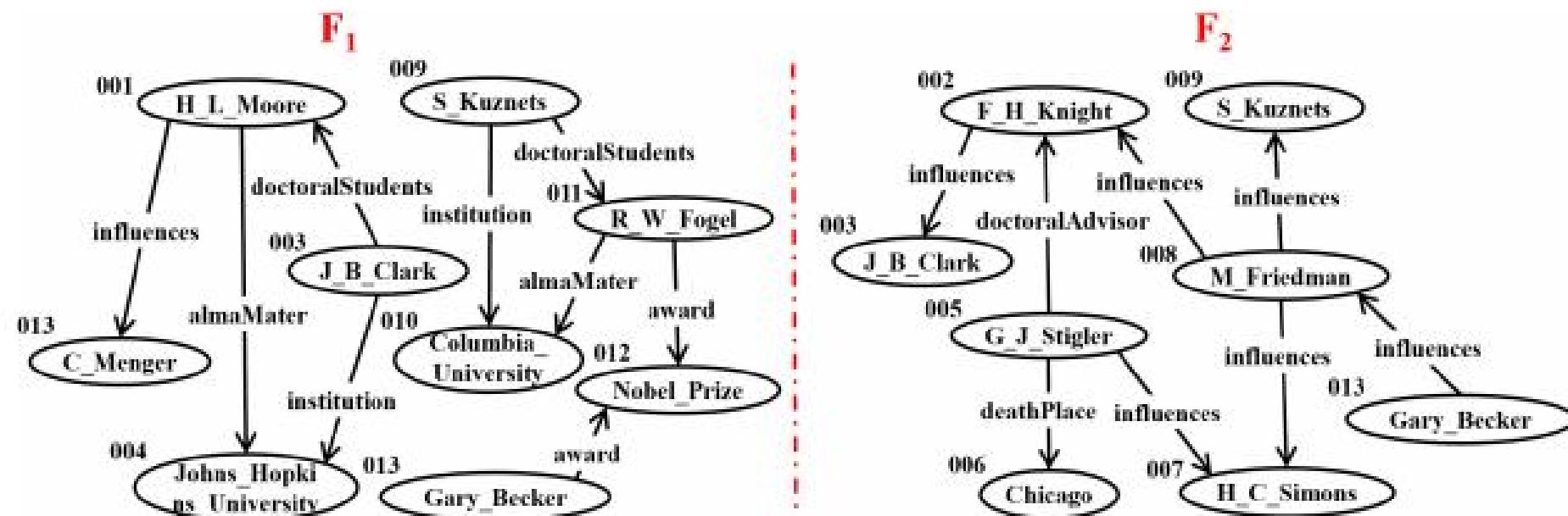
通常， k 取决于分布式系统中机器的数量。

定义2.3 (分区)

给定一个RDF图G，分区F是一组分区 $\{F_1, F_2, \dots, F_k\}$ ，其中每个 $F_i = (V_i, E_i, L_i, f_i)$ ($1 \leq i \leq k$) 满足以下条件：

1. V_i 是分区 F_i 的一组顶点；
2. $\{E_1, \dots, E_k\}$ 是E的一个不相交分区，即 $E_i \subseteq V_i \times V_i$, $E_i \cap E_j = \emptyset$, $1 \leq i, j \leq k$, $i \neq j$, 并且 $\bigcup_{1 \leq i \leq k} E_i = E$ ；
3. L_i 是 F_i 中的一组边标签；
4. $f_i: E_i \rightarrow L_i$ 是一个标签映射，其中对于每个边 $e \in E_i$, 其边标签 $f_i(e)$ 是 $f(e)$ 。

图2(b)显示了对图1(a)中的RDF图的划分。分区中有两个分区， F_1 和 F_2 ，其中 F_1 有9条边， F_2 有8条边。



(b) Example Our Edge-Disjoint Partitioning (MaxLocJoin-EDP)

定义2.4(复杂查询)

复杂查询表示为 $Q = \{V^Q, E^Q, L^Q, f^Q\}$, 其中:

- (1) $V^Q \subseteq V \cup V_{Var}$ 是顶点的集合, 其中 V 表示RDF图 G 中的所有顶点, V_{Var} 是变量的集合;
- (2) $E^Q \subseteq V^Q \times V^Q$ 是 Q 中边的多重集;

定义2.4(复杂查询)

(3) L^Q 是边标签的集合，其中 $L^Q \subseteq L \cup L_{Var} \cup L^* \cup L^+$ ， L 表示 RDF 图 G 中的所有属性， L_{Var} 是属性变量的集合， L^* 和 L^+ 是分别跟有星号(*)或加号(+)的属性集合。这里，一个后跟星号(*)或加号(+)的属性 l 称为对 l 的 **单属性递归约束**，形式化上来说这对应于属性的 Kleene 星号或加号闭包；

(4) $f^Q : E^Q \rightarrow L^Q$ 是一个映射，其中 E^Q 中的每条边 e 都有一个在 L (即属性) 中的边标签 $f^Q(e)$ ，或者边标签是一个变量，或者边标签是一个单属性递归约束

定义2.5(复杂匹配)

考虑一个RDF图G和一个带有n个顶点 $\{v_1, \dots, v_n\}$ 的查询图Q。如果存在一个函数 μ 从 $\{v_1, \dots, v_n\}$ 到 $\{u_1, \dots, u_x\}$ ($n \geq x$)，那么具有x个顶点 $\{u_1, \dots, u_x\}$ (在G中) 的子图被称为Q的匹配，前提是以下条件成立：

- (1) 如果 v_i 不是变量，则 $\mu(v_i) = v_i$ ($1 \leq i \leq n$)；
- (2) 如果 v_i 是变量，则对 $\mu(v_i)$ 没有限制，除了 $\mu(v_i) \in V$ ；

定义2.5(复杂匹配)

- (3) 对于Q中的边 $\overrightarrow{v_i v_j}$, 必须存在G中的边 $\overrightarrow{\mu(v_i)\mu(v_j)}$ 。如果 $f^Q(\overrightarrow{v_i v_j})$ 不是变量, 则 $f^Q(\overrightarrow{v_i v_j}) = f(\overrightarrow{\mu(v_i)\mu(v_j)})$ 。如果 $f^Q(\overrightarrow{v_i v_j})$ 是变量, 则 $\overrightarrow{v_i v_j}$ 可以匹配 $f(\overrightarrow{\mu(v_i)\mu(v_j)})$ 中的任何边属性;
- (4) 对于Q中的边 $\overrightarrow{v_i v_j}$, 如果 $f^Q(\overrightarrow{v_i v_j})$ 是对属性p的单属性递归约束, 则在G中必须有一条从 $\mu(v_i)$ 到 $\mu(v_j)$ 的路径 π , 其中 π 中的每条边都具有属性p。

定义2.6(独立执行)

如果一个SPARQL查询 Q 在RDF图 G 的分区 $F = \{F_1, F_2, \dots, F_k\}$ 上是独立可执行的查询 (IEQ) , 那么它的结果 $MS(Q, G) = \bigcup_{i=1}^k MS(Q, F_i)$, 其中 $MS(Q, F_i)$ 是在分区 F_i 上执行 Q 的匹配集 ($i = 1, \dots, k$) 。

为了解决避免在复杂查询的两个部分中进行跨分区连接的问题，我们首先提出两个关键的观察，然后提出一个基于这些见解的分区方案。

一个有向图 G 被称为弱连通的，如果将其所有有向边替换为无向边后产生一个连通的无向图；

一个RDF图 G 的弱连通分量（WCC） G' 是 G 的一个最大的弱连通子图，使得 G' 不是另一个弱连通分量的子图；

G 的弱连通分量集合表示为 $WCC(G)$ ；

给定一个属性集合 $L' \subseteq L$ 和一个RDF图 G ，由 L' 诱导的 G 的子图，记为 $G[L']$ ，是由仅具有 L' 中属性的边形成的子图。

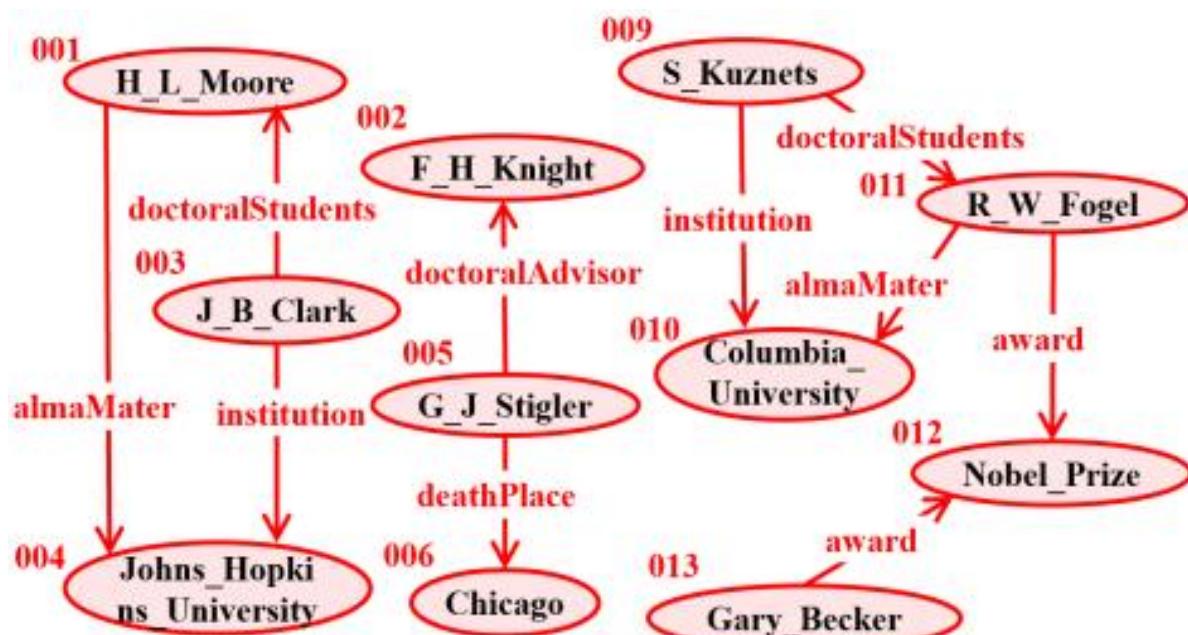
第一个观察：给定一个属性 l ，为了避免针对属性 l 的单属性递归约束进行跨分区连接，我们只需要确保由具有属性 l 的所有边诱导的子图($G[\{l\}]$)的每个弱连通分量内的所有边都在同一个分区中。

我们不需要将所有具有属性 l 的边放在一个分区中，这在大多数边不相交的分区中是常见的。

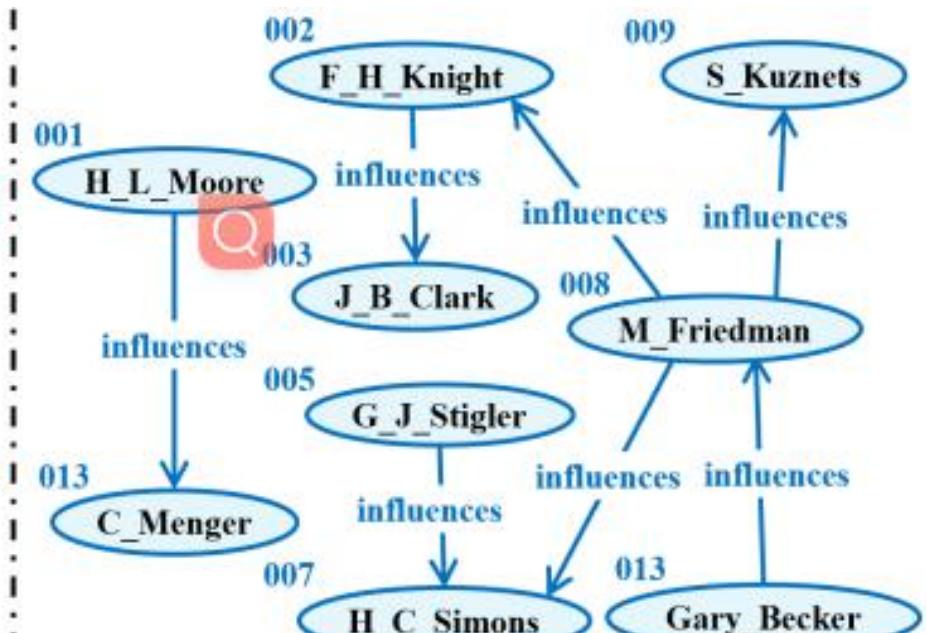
第二个观察：在RDF图的分区中，如果存在一组属性 L_{local} ，使得由具有 L_{local} 中属性的所有边诱导的子图($G[L_{local}]$)的每个弱连通分量都在一个单独的分区中，那么只涉及 L_{local} 中属性的查询可以是独立可执行的。

我们将 L_{local} 中的属性称为**局部可连接属性**。那么，很明显，局部可连接属性的集合越大，没有单属性递归约束的查询成为IEQ的可能性就越高。

$G[L_{local}]$ 中的所有弱连接组件都包含在单独的分区中。



$G[L_{local}]$, where $L_{local}=\{almaMater, doctoralStudents, institution, deathplace, doctoralAdvisor, award\}$



$G[L - L_{local}]$, where $L - L_{local}=\{\text{influences}\}$

定理1

给定一个相对于分区F的局部可连接属性集 L_{local} ，只包含 L_{local} 中属性的查询是可以独立执行的。

证明：考虑一个只包含 L_{local} 中属性的查询Q。对于Q的任何匹配，其所有边必须相连。因此，Q的任何匹配都不能跨越 $G[L_{local}]$ 中的多个弱连通分量。根据局部可连接属性的定义，由于 $G[L_{local}]$ 中的任何弱连通分量都不能跨越F中的多个分区，因此Q的任何匹配都不能跨越F中的多个分区。然后，根据定义2.6，Q是关于分区F的IEQ。

定义3.1(MaxLocJoin-EDP)

给定一个RDF图G和一个正整数k，G的MaxLocJoin-EDP是一个分区 $F = \{F_1, F_2, \dots, F_k\}$ ，满足以下条件：

- (1) 对于任何属性l， $G[\{l\}]$ 中的每个弱连通分量都不会跨越多个分区；
- (2) 对于每个 F_i ， F_i 的大小（即 $|E_i|$ ）不大于 $(1+\varepsilon) \times |E|/k$ ，其中 ε 是用户定义的最大不平衡比率（即分区相对大小可以有多大的差异）；
- (3) 局部可连接属性集的大小 $|L_{local}|$ 最大化

定理2

MaxLocJoin-EDP问题是NP-完全的。

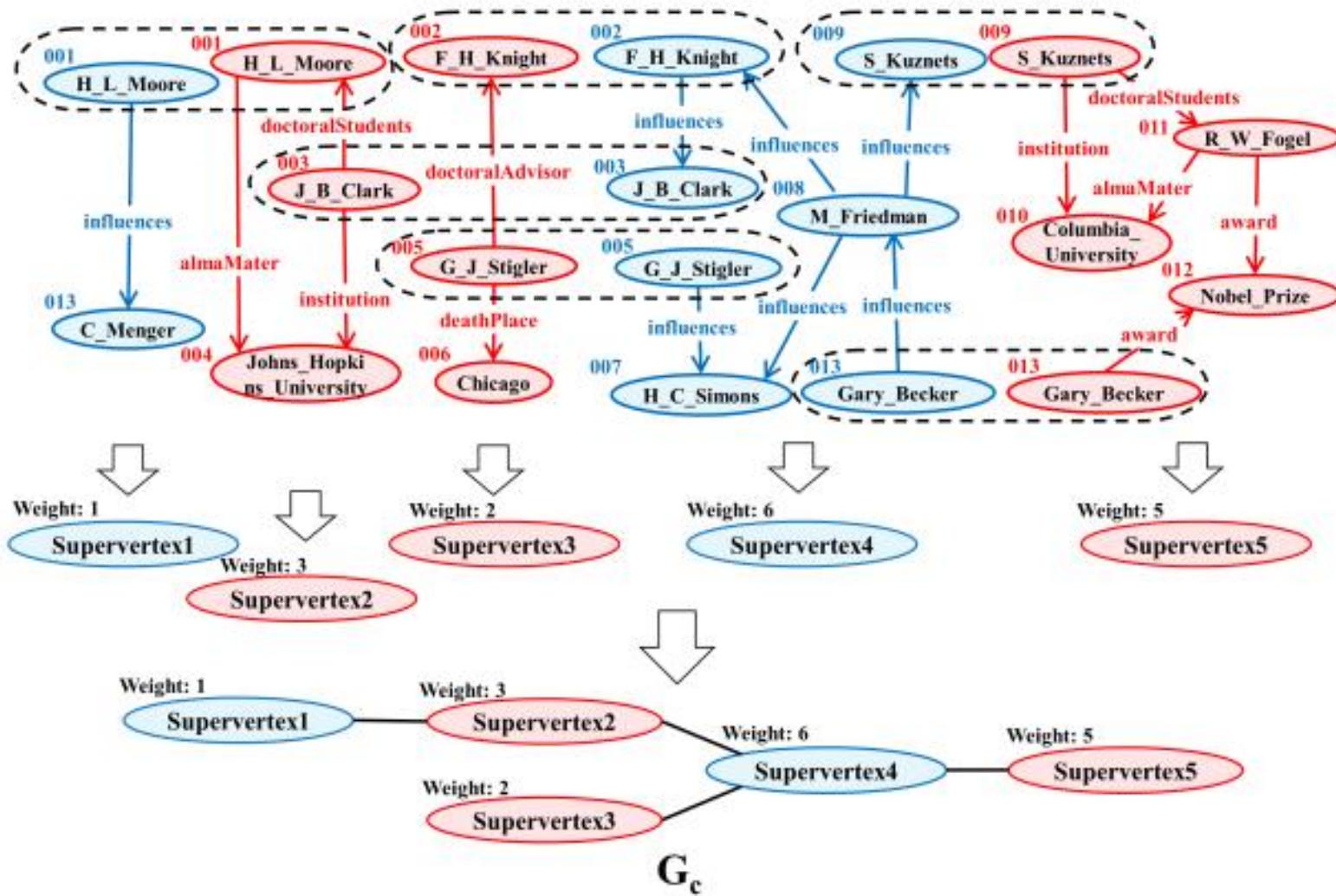
证明：集合划分问题(SPP)已被证明是NP-完全的。为了证明MaxLocJoin-EDP问题是NP-完全的，我们将SPP的一个实例简化为MaxLocJoin-EDP的一个实例。设 $\varphi = \{x_1, x_2, \dots, x_{|\varphi|}\}$ 是一个正整数的输入集合，集合划分问题的目标是找到两个不相交的互补子集 φ_1 和 φ_2 ，使得 φ_1 中元素的和等于 φ_2 中元素的和。

贪婪算法：找到满足要求的局部可连接属性集 L_{local}

设 $G[L_{local}]$ 是由具有 L_{local} 中属性的边诱导的子图，设 $L' = L - L_{local}$ 是不在 L_{local} 中的属性集。我们将 $G[\{l_1\}]$, $G[\{l_2\}]$, ..., $G[\{l_t\}]$ 定义为 L' 中每个属性诱导的子图，其中 t 表示 L' 中属性的数量。然后，我们将每个弱连通分量 (WCC) 应用到 $G[L_{local}]$ 、 $G[\{l_1\}]$ 、 $G[\{l_2\}]$ 、...、 $G[\{l_t\}]$ 的粗化中，将每个 WCC 视为一个超级顶点

这样，我们得到了一个粗化图 G_c ，如果两个超级顶点之间有边，那么它们对应的 WCC 中有重叠的顶点。 G_c 中的顶点数远小于 G 中的顶点数， G_c 中的每个顶点代表一个 WCC

分区框架



定义3.2（选择局部可连接属性成本）

给定一组属性 $L' \subseteq L$ ，选择 L' 作为局部可连接属性集的成本为：

$$\text{Cost}(L') = \max_{c \in WCC(G[L'])} |c|$$

其中， $WCC(G[L'])$ 是 $G[L']$ 中弱连通分量的集合， c 是 $WCC(G[L'])$ 中的一个弱连通分量， $|c|$ 表示 c 中边的数量。

另一方面，根据定义3.1，MaxLocJoin-EDP要求每个分区中的边的数量不应超过 $(1 + \varepsilon) \times |E|/k$ 。因此，我们使用定义3.2中的成本函数来选择尽可能多的属性形成局部可连接属性集，前提是所选集的成本不超过 $(1 + \varepsilon) \times |E|/k$ 。

因此，我们在算法1中提出了一种成本驱动的贪婪算法来选择局部可连接属性，其目标是在定义3.1中定义的分区大小约束下最大化局部可连接属性的数量。该算法在每个迭代中选择对局部可连接属性集增量成本最小的属性。

具体来说，该算法可以分为两部分。

首先，局部可连接属性集 L_{local} 初始化为空，我们计算每个属性诱导的子图的 WCCs 以及每个属性的成本（行 1-4）。

Algorithm 1: Locally Joinable Property Selection Algorithm

Input: An RDF graph $G = (V, E, L, f)$

Output: A set of locally joinable property $L_{local} \subseteq L$

```
1  $L_{local} \leftarrow \emptyset;$ 
2 for each property  $l$  in  $L$  do
3   Compute  $WCC(G[\{l\}])$ , i.e., weakly connected components
      in  $G[\{l\}]$ ;
4   Compute the cost of  $Cost(\{l\})$  according to Definition 3.2.
5 while  $L \neq \emptyset$  do
6    $mincost \leftarrow \infty; l_{opt} \leftarrow \phi$ 
7   for each property  $l$  in  $L$  do
8     Compute  $WCC(G[L_{local} \cup \{l\}])$ 
9     if  $Cost(L_{local} \cup \{l\}) < (1 + \epsilon) \times |E|/k$  then
10       if  $Cost(L_{local} \cup \{l\}) < mincost$  then
11          $mincost \leftarrow Cost(L_{local} \cup \{l\});$ 
12          $l_{opt} \leftarrow l$ 
13       if  $l_{opt} == \phi$  then
14         BREAK
15      $L_{local} \leftarrow L_{local} \cup \{l_{opt}\}; L \leftarrow L - \{l_{opt}\};$ 
16 Return  $L_{local};$ 
```

然后，在每个迭代中，通过计算不在 L_{local} 中的每个属性 l 的成本 $Cost(L_{local} \cup \{l\})$ 来确定最优属性 l_{opt} （行7-12）。

Algorithm 1: Locally Joinable Property Selection Algorithm

Input: An RDF graph $G = (V, E, L, f)$

Output: A set of locally joinable property $L_{local} \subseteq L$

```
1  $L_{local} \leftarrow \emptyset;$ 
2 for each property  $l$  in  $L$  do
3   Compute  $WCC(G[\{l\}])$ , i.e., weakly connected components
   in  $G[\{l\}]$ ;
4   Compute the cost of  $Cost(\{l\})$  according to Definition 3.2.
5 while  $L \neq \emptyset$  do
6    $mincost \leftarrow \infty; l_{opt} \leftarrow \phi$ 
7   for each property  $l$  in  $L$  do
8     Compute  $WCC(G[L_{local} \cup \{l\}])$ 
9     if  $Cost(L_{local} \cup \{l\}) < (1 + \epsilon) \times |E|/k$  then
10       if  $Cost(L_{local} \cup \{l\}) < mincost$  then
11          $mincost \leftarrow Cost(L_{local} \cup \{l\});$ 
12          $l_{opt} \leftarrow l$ 
13       if  $l_{opt} == \phi$  then
14         BREAK
15      $L_{local} \leftarrow L_{local} \cup \{l_{opt}\}; L \leftarrow L - \{l_{opt}\};$ 
16 Return  $L_{local};$ 
```

然后将属性 l_{opt} 插入到 L_{local} 并从 L 中删除（行 15）。该算法重复这些步骤，直到 L 为空（行 5）或因成本约束而无法选择更多属性为止（行 13-14）。

Algorithm 1: Locally Joinable Property Selection Algorithm

Input: An RDF graph $G = (V, E, L, f)$

Output: A set of locally joinable property $L_{local} \subseteq L$

```
1  $L_{local} \leftarrow \emptyset;$ 
2 for each property  $l$  in  $L$  do
3   Compute  $WCC(G[\{l\}])$ , i.e., weakly connected components
    in  $G[\{l\}]$ ;
4   Compute the cost of  $Cost(\{l\})$  according to Definition 3.2.
5 while  $L \neq \emptyset$  do
6    $mincost \leftarrow \infty; l_{opt} \leftarrow \phi$ 
7   for each property  $l$  in  $L$  do
8     Compute  $WCC(G[L_{local} \cup \{l\}])$ 
9     if  $Cost(L_{local} \cup \{l\}) < (1 + \epsilon) \times |E|/k$  then
10      if  $Cost(L_{local} \cup \{l\}) < mincost$  then
11         $mincost \leftarrow Cost(L_{local} \cup \{l\});$ 
12         $l_{opt} \leftarrow l$ 
13      if  $l_{opt} == \phi$  then
14        BREAK
15       $L_{local} \leftarrow L_{local} \cup \{l_{opt}\}; L \leftarrow L - \{l_{opt}\};$ 
16 Return  $L_{local};$ 
```

最后，该算法返回选择的局部可连接属性集 L_{local} （行 16）。

Algorithm 1: Locally Joinable Property Selection Algorithm

Input: An RDF graph $G = (V, E, L, f)$
Output: A set of locally joinable property $L_{local} \subseteq L$

```
1  $L_{local} \leftarrow \emptyset;$ 
2 for each property  $l$  in  $L$  do
3   Compute  $WCC(G[\{l\}])$ , i.e., weakly connected components
   in  $G[\{l\}]$ ;
4   Compute the cost of  $Cost(\{l\})$  according to Definition 3.2.
5 while  $L \neq \emptyset$  do
6    $mincost \leftarrow \infty; l_{opt} \leftarrow \phi$ 
7   for each property  $l$  in  $L$  do
8     Compute  $WCC(G[L_{local} \cup \{l\}])$ 
9     if  $Cost(L_{local} \cup \{l\}) < (1 + \epsilon) \times |E|/k$  then
10       if  $Cost(L_{local} \cup \{l\}) < mincost$  then
11          $mincost \leftarrow Cost(L_{local} \cup \{l\});$ 
12          $l_{opt} \leftarrow l$ 
13       if  $l_{opt} == \phi$  then
14         BREAK
15      $L_{local} \leftarrow L_{local} \cup \{l_{opt}\}; L \leftarrow L - \{l_{opt}\};$ 
16   Return  $L_{local};$ 
```

算法1的瓶颈在第3行和第8行，即计算WCCs。第一步是计算 $G[\{l\}]$ 中的WCCs(第3行)。第二步是迭代合并 $G[L_{local}]$ 和 $G[\{l\}]$ 中的WCCs(第8行)。

Algorithm 1: Locally Joinable Property Selection Algorithm

Input: An RDF graph $G = (V, E, L, f)$
Output: A set of locally joinable property $L_{local} \subseteq L$

```

1    $L_{local} \leftarrow \emptyset;$ 
2   for each property  $l$  in  $L$  do
3       Compute  $WCC(G[\{l\}])$ , i.e., weakly connected components
      in  $G[\{l\}]$ ;
4       Compute the cost of  $Cost(\{l\})$  according to Definition 3.2.
5   while  $L \neq \emptyset$  do
6        $mincost \leftarrow \infty; l_{opt} \leftarrow \phi$ 
7       for each property  $l$  in  $L$  do
8           Compute  $WCC(G[L_{local} \cup \{l\}])$ 
9           if  $Cost(L_{local} \cup \{l\}) < (1 + \epsilon) \times |E|/k$  then
10              if  $Cost(L_{local} \cup \{l\}) < mincost$  then
11                   $mincost \leftarrow Cost(L_{local} \cup \{l\});$ 
12                   $l_{opt} \leftarrow l$ 
13              if  $l_{opt} == \phi$  then
14                  BREAK
15               $L_{local} \leftarrow L_{local} \cup \{l_{opt}\}; L \leftarrow L - \{l_{opt}\};$ 
16   Return  $L_{local}$ ;

```

主要取决于WCC的计算（第3行和第8行），这可以通过使用不相交集合森林来实现

构建不相交集合森林的复杂性是 $O(\alpha(|V|) \times |E|)$ ，其中 $\alpha(|V|)$ 是逆Ackermann函数，对于所有实际目的都小于5

合并两个不相交集合森林需要 $O(\alpha(|V|) \times |E|)$ 时间。考虑到算法1第5-7行的双重循环，时间复杂度是 $O(|L|^2 \times \alpha(|V|) \times |E|)$ 。

因此，算法1的总时间复杂度是 $O(|L|^2 \times \alpha(|V|) \times |E|)$ 。通常，属性的数量远小于边的数量（即 $|L| \ll |E|$ ）， $\alpha(|V|)$ 是一个微不足道的值，所以复杂性主要取决于边的数量， $|E|$ 。

定义4.1 (局部可连接的可独立执行查询)

如果查询中所有边的属性都在本地可连接的属性集(L_{local})中，则查询 Q 是一个局部可连接的可独立执行查询。

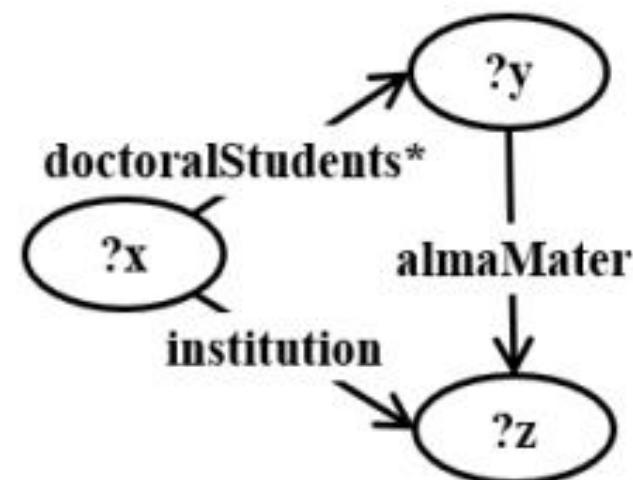
定理3

一个局部可连接的可独立执行查询可以在没有分区间连接的情况下执行。

证明：如果一个局部可连接的IEQ Q 有一个匹配 μ 并且需要分区间连接，那么在 μ 中至少有两个相邻的边 e_1 和 e_2 在不同的分区中。然而，由于边 e_1 和 e_2 具有 L_{local} 属性，不同分区中的边不能在同一弱连通分量中，因此边 e_1 和 e_2 应该是不连通的。这与 e_1 和 e_2 是相邻的事实相矛盾。

本地可独立执行的查询

图5(a)是一个局部可连接的IEQ，因为它包含的所有属性都在 L_{local} 中。查询可以在分区 F_1 和 F_2 上独立执行，它的所有匹配项都在 F_1 中。

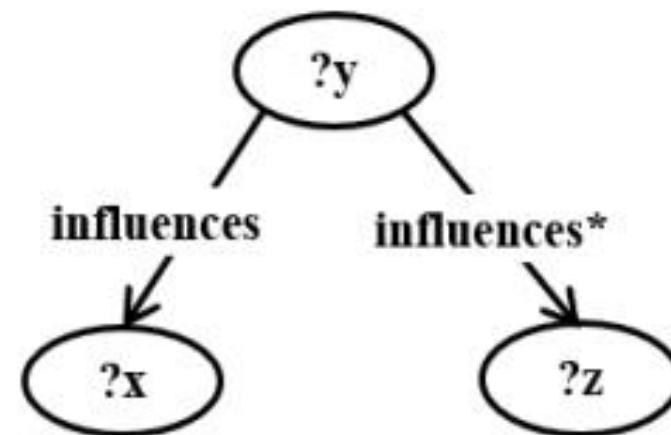


(a) Example Locally Joinable IEQ

定义4.2（单属性可独立执行查询）

只有一个属性的SPARQL查询Q是单属性独立可执行查询。

图5(b)中显示的查询是单属性IEQ的一个例子，因为它只包含属性 $\langle e:1 \rangle influences$ ，而它不在本地可连接的属性集 L_{local} 中。很明显，这个查询的所有匹配都在分区 F_2 中。



(b) Example Single-Property IEQ

给定一个复杂查询 Q 和一组本地可连接属性集 L_{local} , 算法2展示了如何获得分解后的独立可执行子查询的集合 Q

算法扫描 E^Q 一个接一个地把查询图 Q 分成两部分, Q' 和 Q'' , Q' 包含边缘与属性 L_{local} 和 Q'' 与属性包含边缘 $L - L_{local}$ (1-5行)

Algorithm 2: Query Decomposition Algorithm

Input: A SPARQL Complex Query $Q = \{V^Q, E^Q, L^Q, f^Q\}$, and a PCEP partitioning \mathcal{F} with the set of locally joinable properties L_{local}

Output: A query decomposition result Q

```
1 for each edge  $e$  in  $E^Q$  do
2   if  $f^Q(e) \in L_{local}$  then
3     Add  $e$  to  $Q'$ ;
4   else if  $f^Q(e) \notin L_{local}$  then
5     Add  $e$  to  $Q''$ ;
6   Compute the weakly connected components in  $Q'$ ;
7   Compute the weak connection components of each property in  $Q''$ 
      separately ;
8 for each weakly connected component  $q$  in  $Q'$  or  $Q''$  do
9   Add  $q$  in  $Q$ ;
10 Return  $Q$ ;
```

然后算法计算 Q' 和 Q'' 中的所有弱连接分量(第6行)，这也可以通过前文中提到的不相交集合森林技术来实现。

Algorithm 2: Query Decomposition Algorithm

Input: A SPARQL Complex Query $Q = \{V^Q, E^Q, L^Q, f^Q\}$, and a PCEP partitioning \mathcal{F} with the set of locally joinable properties L_{local}

Output: A query decomposition result Q

```
1 for each edge  $e$  in  $E^Q$  do
2   if  $f^Q(e) \in L_{local}$  then
3     Add  $e$  to  $Q'$ ;
4   else if  $f^Q(e) \notin L_{local}$  then
5     Add  $e$  to  $Q''$ ;
6   Compute the weakly connected components in  $Q'$ ;
7   Compute the weak connection components of each property in  $Q''$ 
      separately ;
8 for each weakly connected component  $q$  in  $Q'$  or  $Q''$  do
9   Add  $q$  in  $Q$ ;
10 Return  $Q$ ;
```

最后，每个弱连接组件映射到一个子查询，并添加到分解结果 Q 中(第7-8行)。

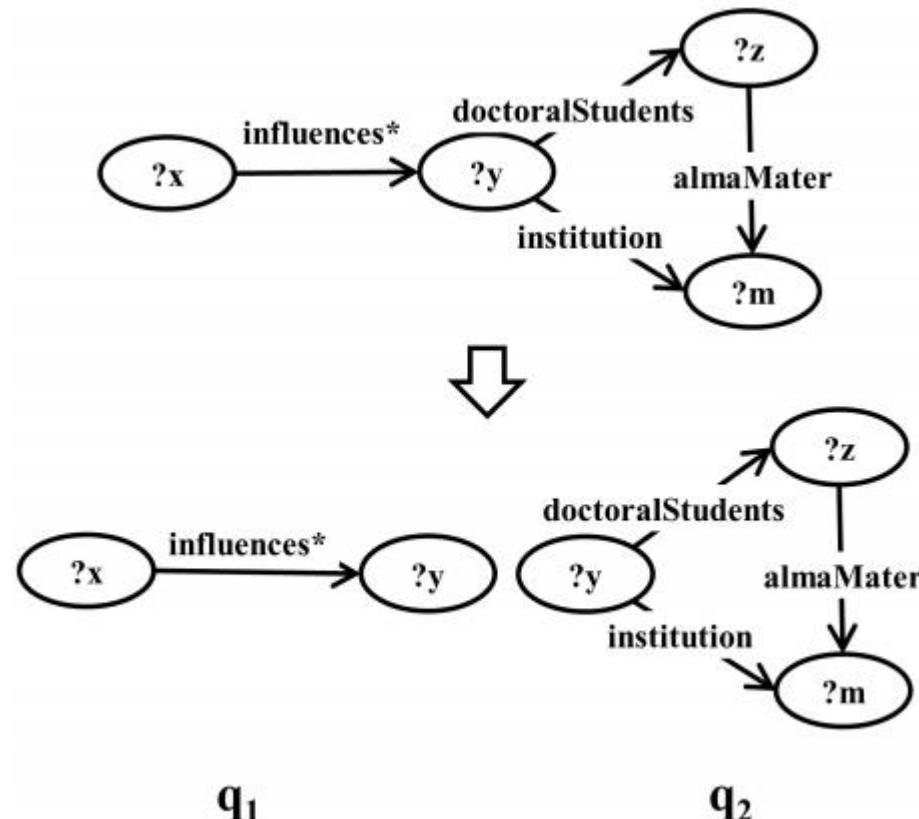
Algorithm 2: Query Decomposition Algorithm

Input: A SPARQL Complex Query $Q = \{V^Q, E^Q, L^Q, f^Q\}$, and a PCEP partitioning \mathcal{F} with the set of locally joinable properties L_{local}

Output: A query decomposition result Q

```
1 for each edge  $e$  in  $E^Q$  do
2   if  $f^Q(e) \in L_{local}$  then
3     Add  $e$  to  $Q'$ ;
4   else if  $f^Q(e) \notin L_{local}$  then
5     Add  $e$  to  $Q''$ ;
6   Compute the weakly connected components in  $Q'$ ;
7   Compute the weak connection components of each property in  $Q''$ 
      separately ;
8 for each weakly connected component  $q$  in  $Q'$  or  $Q''$  do
9   Add  $q$  in  $Q$ ;
10 Return  $Q$ ;
```

在算法2的第一次扫描后，三元模式 $\langle ?x, \text{inflations}^*, ?y \rangle$ 被包含在 Q'' 中，其余的边都在 Q' 中，因为只有属性 inflations 不在 L_{local} 中。 Q'' 和 Q' 都只有一个弱连通分量，所以它们分别映射到一个子查询 q_1 和 q_2 。最后，查询分解结果是 $\{q_1, q_2\}$ 并返回。



□ 数据集：

Dataset	#Triples	#Entities	#Properties
DBpedia 100M	103,677,659	19,027,155	52,603
DBpedia 500M	512,213,528	65,588,189	73,444
DBpedia 1B	1,111,481,066	139,493,254	124,034
Wikidata	2,730,469,540	285,496,189	10,117

□ 竞争对手：Metis、MPC

□ 环境：在阿里云上运行Linux的8台机器

□ 本地可连接属性数目

Dataset	MaxLocJoin-EDP		
	$ L_{local} $	$ L - L_{local} $	$ L_{local} /L$
DBpedia 100M	52,551	52	99.90%
DBpedia 500M	73,398	46	99.94%
DBpedia 1B	123,981	53	99.96%
Wikidata	10,044	73	99.27%

□ DBpedia上分解的子查询数

	MaxLocJoin-EDP	VP	Metis	MPC
CQ_1	1	2	2	2
CQ_2	1	4	2	1
CQ_3	2	2	2	2
CQ_4	1	2	2	2
GQ_1	2	3	1	1
GQ_2	1	2	1	1
GQ_3	2	5	2	1
GQ_4	2	3	3	2

□ 维基数据上分解子查询的平均数目

	MaxLocJoin-EDP	VP	Metis	MPC
Complex Queries	2.2	2.8	2	2
General Queries	1.6	2.4	1.6	1.2

□ 基于维基数据的各阶段评价(单位:毫秒)

	Average Query Response Times	
	Complex Queries	General Queries
QDT	94	81
LET	932,030	9,636
JT	330	17
Total	932,454	9,734

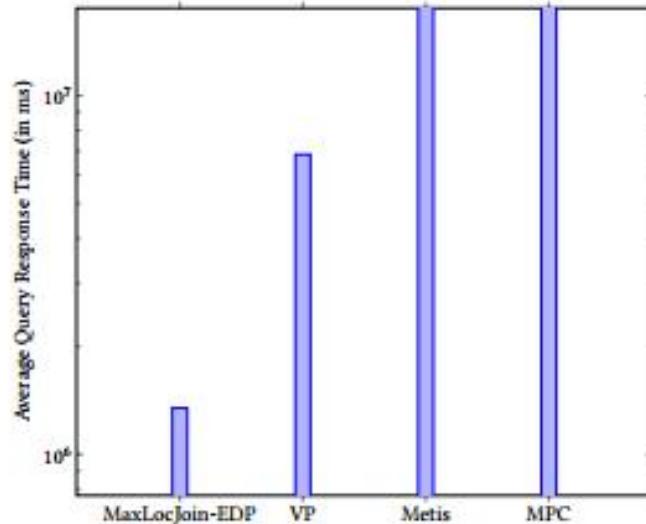
□ 不同RDF图大小的分解子查询的数量

	100M	500M	1B
CQ_1	1	1	1
CQ_2	1	1	1
CQ_3	2	1	1
CQ_4	1	1	1
GQ_1	2	2	1
GQ_2	1	1	1
GQ_3	2	2	2
GQ_4	2	2	2

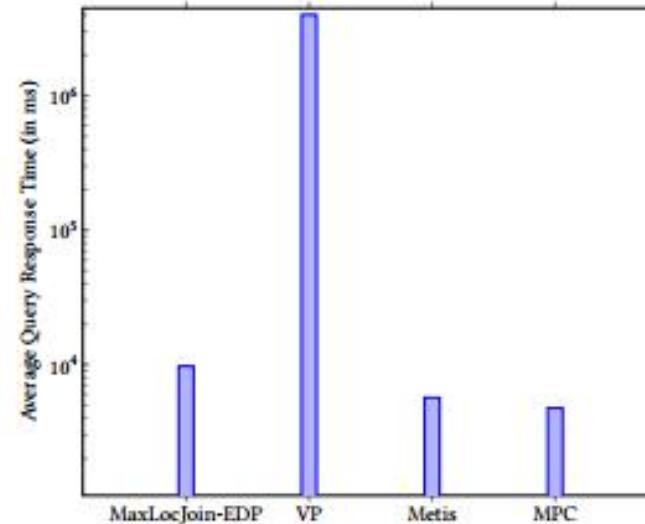
□ 基于维基数据的各阶段评价(单位:毫秒)

Partition Number	4	6	8	10
$ L_{local} $	52,574	52,556	52,551	52,543

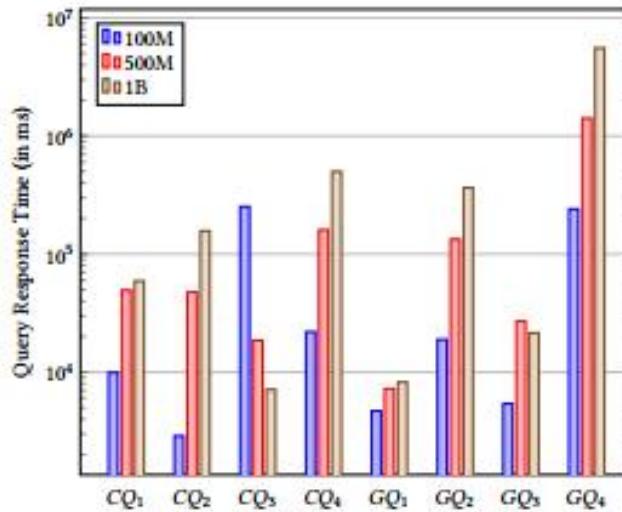
□ 基于维基数据的在线性能比较



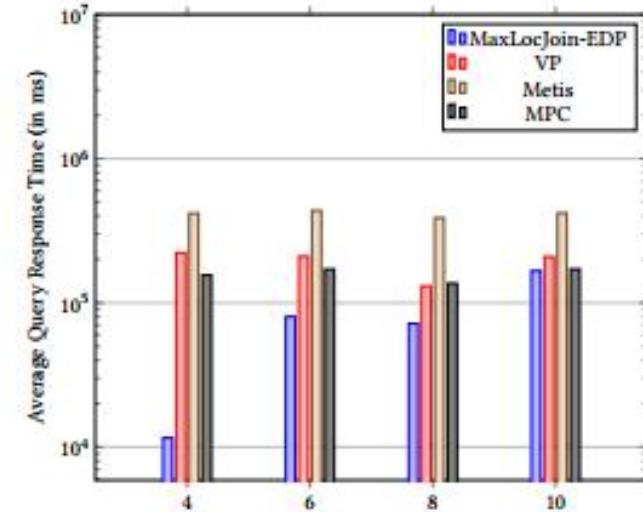
(a) Complex Queries



(b) General Queries



(a) Query Performance Varying RDF Graph Sizes



(b) Complex Query Performance Varying Number of Partitions

□ 分区和加载时间(分钟)

Strategies	Partitioning	Loading	Total
MaxLocJoin-EDP	2,419	303	2,722
VP	334	448	782
Metis	620	868	1,488
MPC	1145	358	1,503



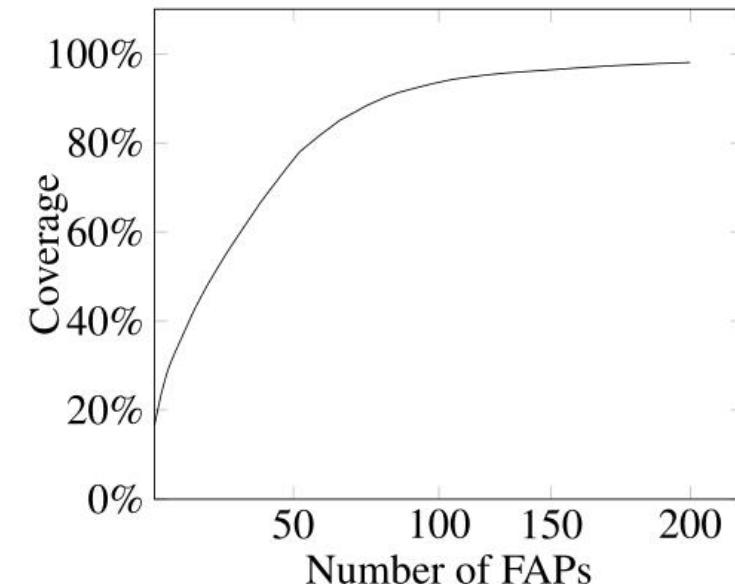
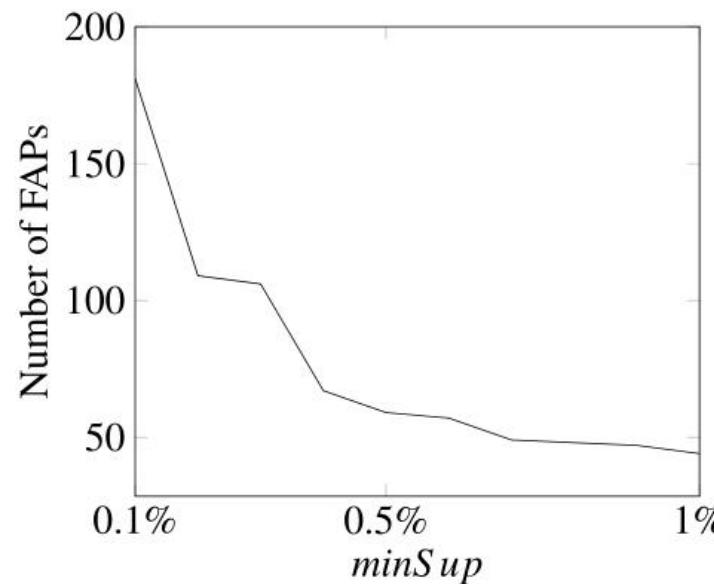
第三部分

基于查询日志 的方法



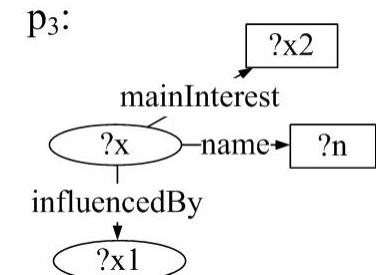
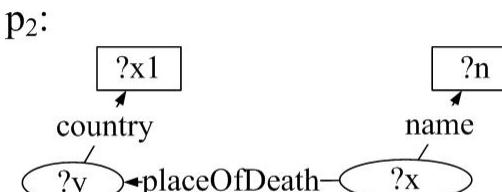
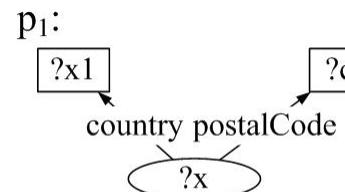
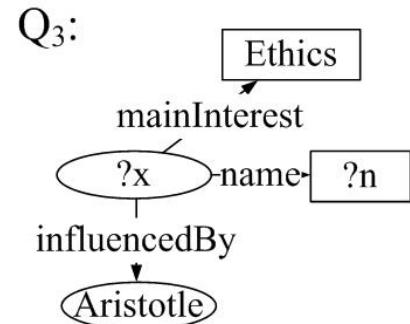
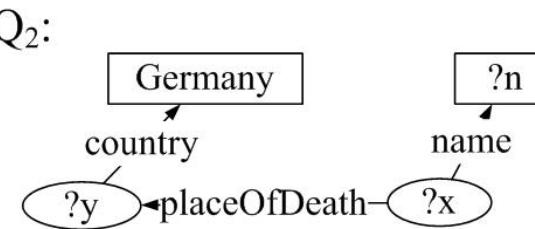
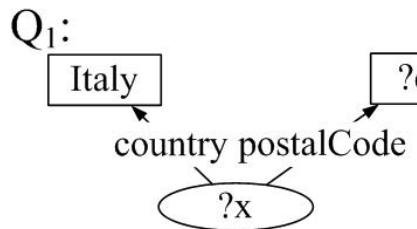
为什么查询工作负载很重要？

一些模式可以覆盖实际查询日志中的大多数查询。



查询日志的规范化

在进行频繁访问模式挖掘之前，我们规范化工作负载中的所有 SPARQL 查询，以避免过拟合。



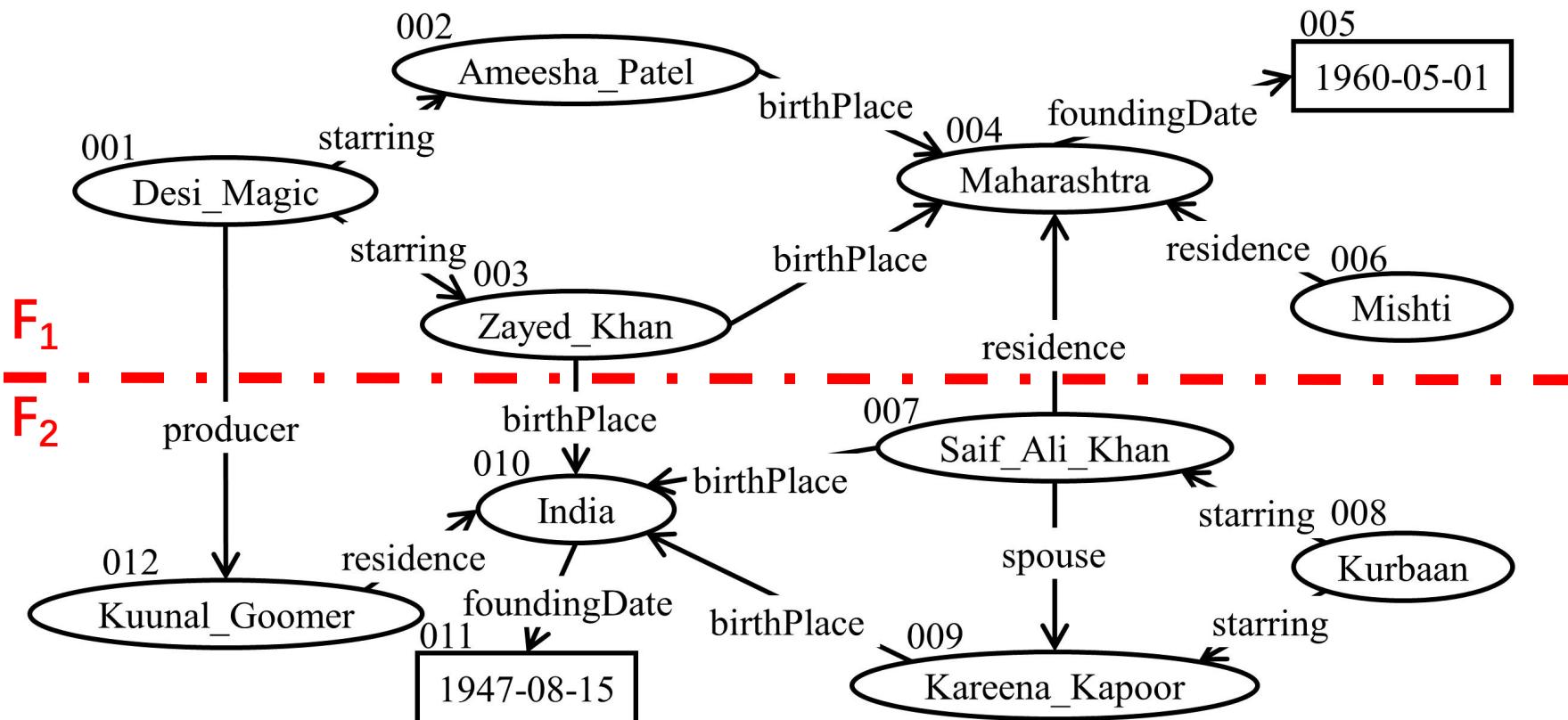
访问频率

- 模式p的访问频率是工作负载Q中包含模式p作为子图的查询的数量。
- 如果模式p的访问频率不低于阈值minSup，则模式p是频繁访问模式。
- 大多数与工作负载有关的方法都基于频繁访问模式。

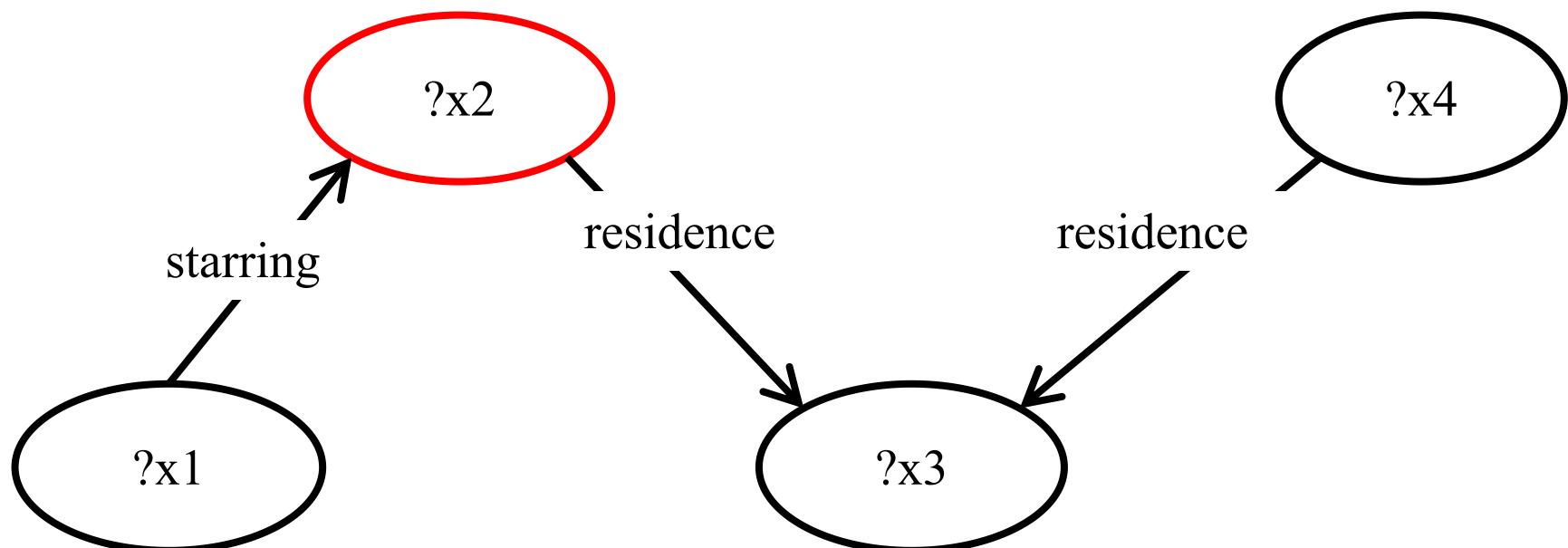
WARP提出了一种分布式SPARQL引擎，
它将图分割技术与跨分区的三元组的工作负载感知复制相结合。



首先，WARP使用METIS对RDF图进行分区。

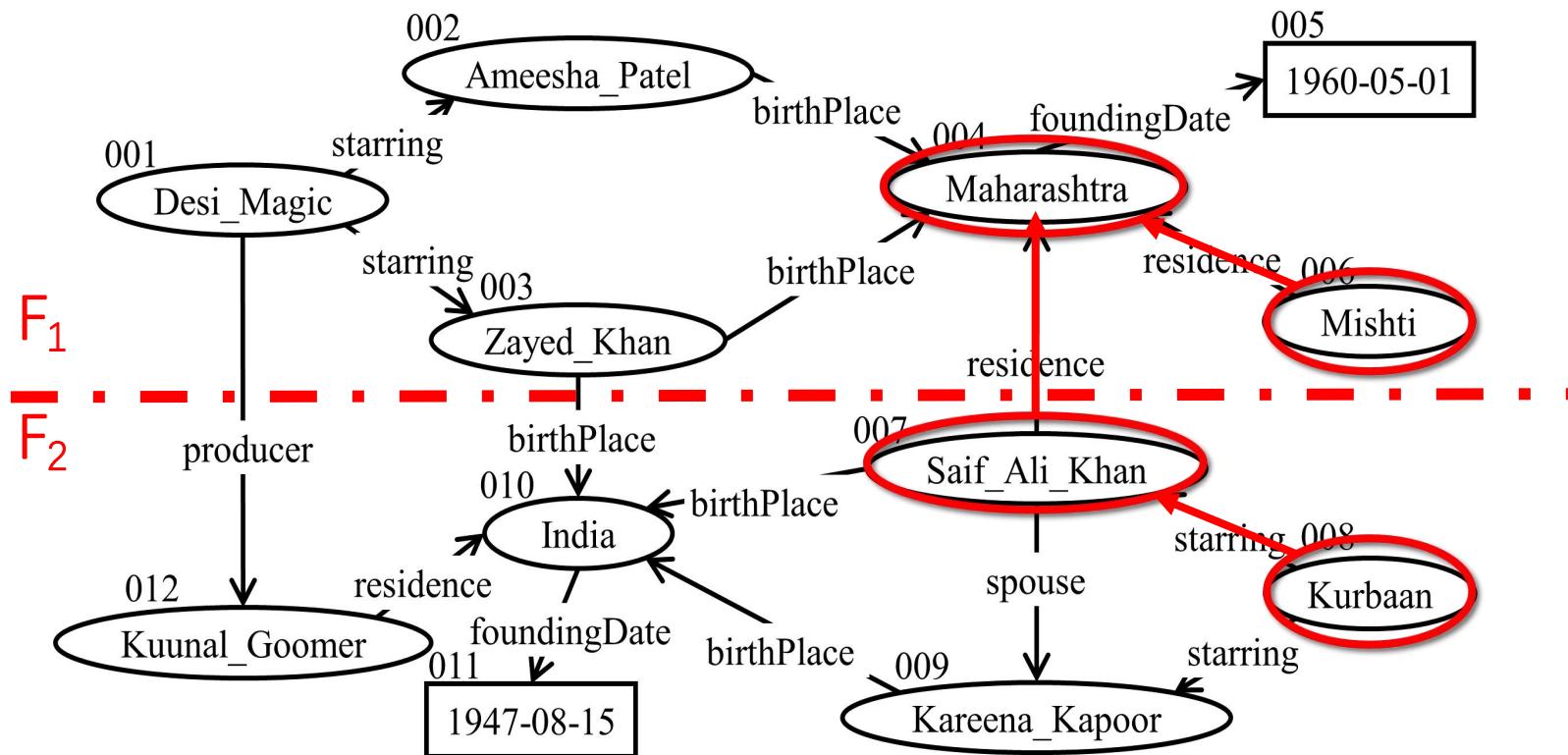


其次，WARP找到一些无法独立执行的频繁查询模式，并选择一个查询顶点作为种子。



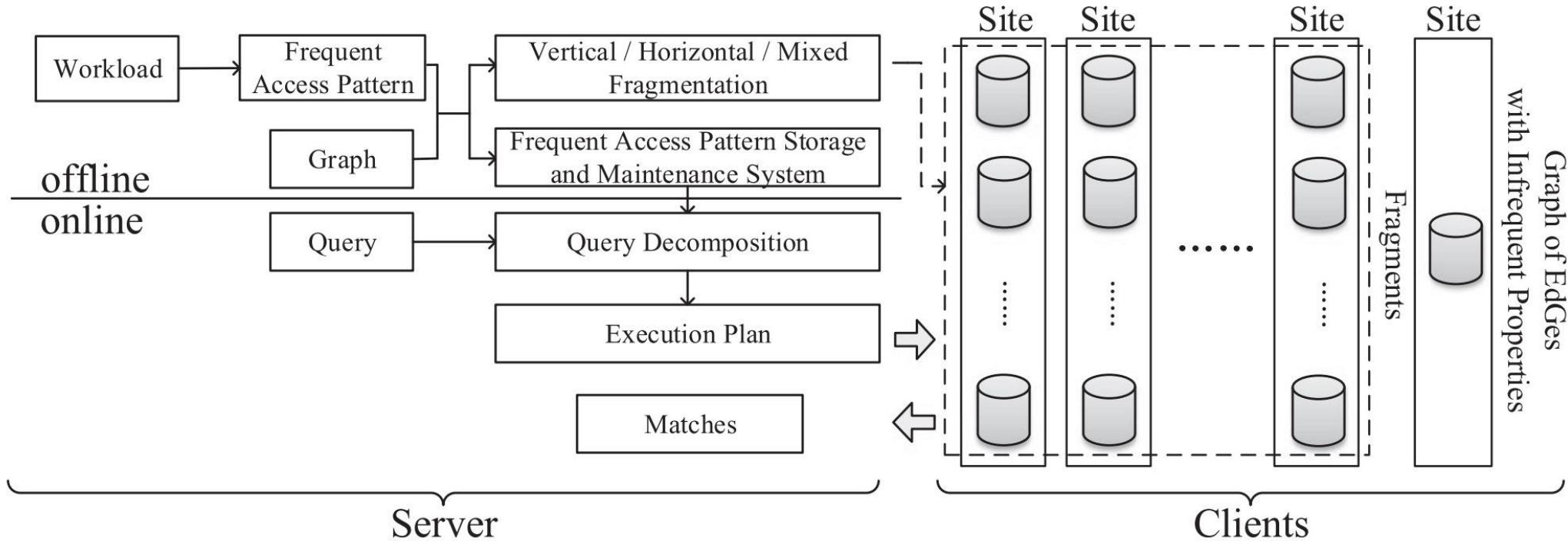
最后，WARP在种子映射所在的分区上复制模式的匹配项。

种子顶点?x2映射到007，因此该匹配项在F2上进行复制。



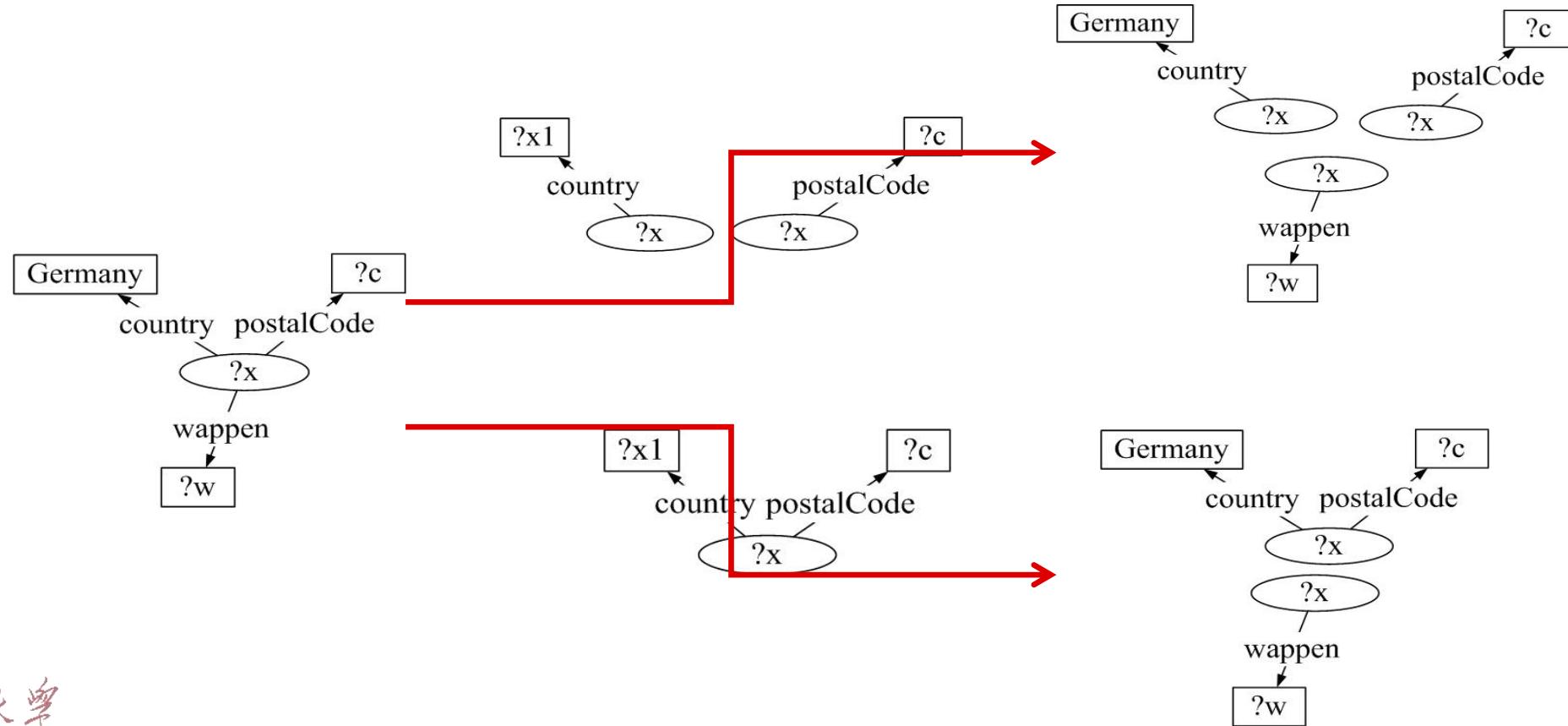
□ 优点：如果一个查询与频繁查询模式同构，那么就不需要进行分区间连接，查询性能非常高。

- 首先，我们在查询工作负载中挖掘和选择频繁子图模式，称为**频繁访问模式**。
- 我们提出了三种基于这些模式的碎片化策略，即**垂直、水平和混合碎片化**，以及一种分配算法，用于将分区分布到分布式系统中的站点。
- 我们提出了一种成本感知的查询优化方法来评估SPARQL查询。



频繁访问模式的选择

- 并非所有模式都需要被选中
- 较大的模式具有较大的好处



- 为命中查询Q而选择频繁访问模式p的好处是

$$Benefit(p, Q) = |E(p)| \times use(Q, p)$$

其中 $use(Q, p) = 1$ 如果模式p是Q的子图

- 给定一组在工作负载Q上的频繁访问模式P，其好处是

$$Benefit(p, Q) = \sum_{Q_i \in Q, i=1, \dots, r} \max_{p \in P} \{ Benefit(p, Q_i) \}$$

- 我们将存储约束归一化为一个值SC

$$\sum_{p \in P'} |E(\llbracket p \rrbracket_G)| \leq SC$$

- 其中 $E(\llbracket p \rrbracket_G)$ 表示G中与p同构的所有子图匹配的大小

- 在受到存储约束的情况下，找到一个具有最大好处的频繁访问模式集是NP-hard的
- 因此，我们提出了一种启发式算法，它可以保证数据完整性

选择所有具有一条边的模式以保证数据完整性

贪婪选择频繁访问模式

近似比为 $\min\left\{\frac{1}{\max_{p \in P}|E(p)|}, \frac{1}{2}\left(1 - \frac{1}{e}\right)\right\}$

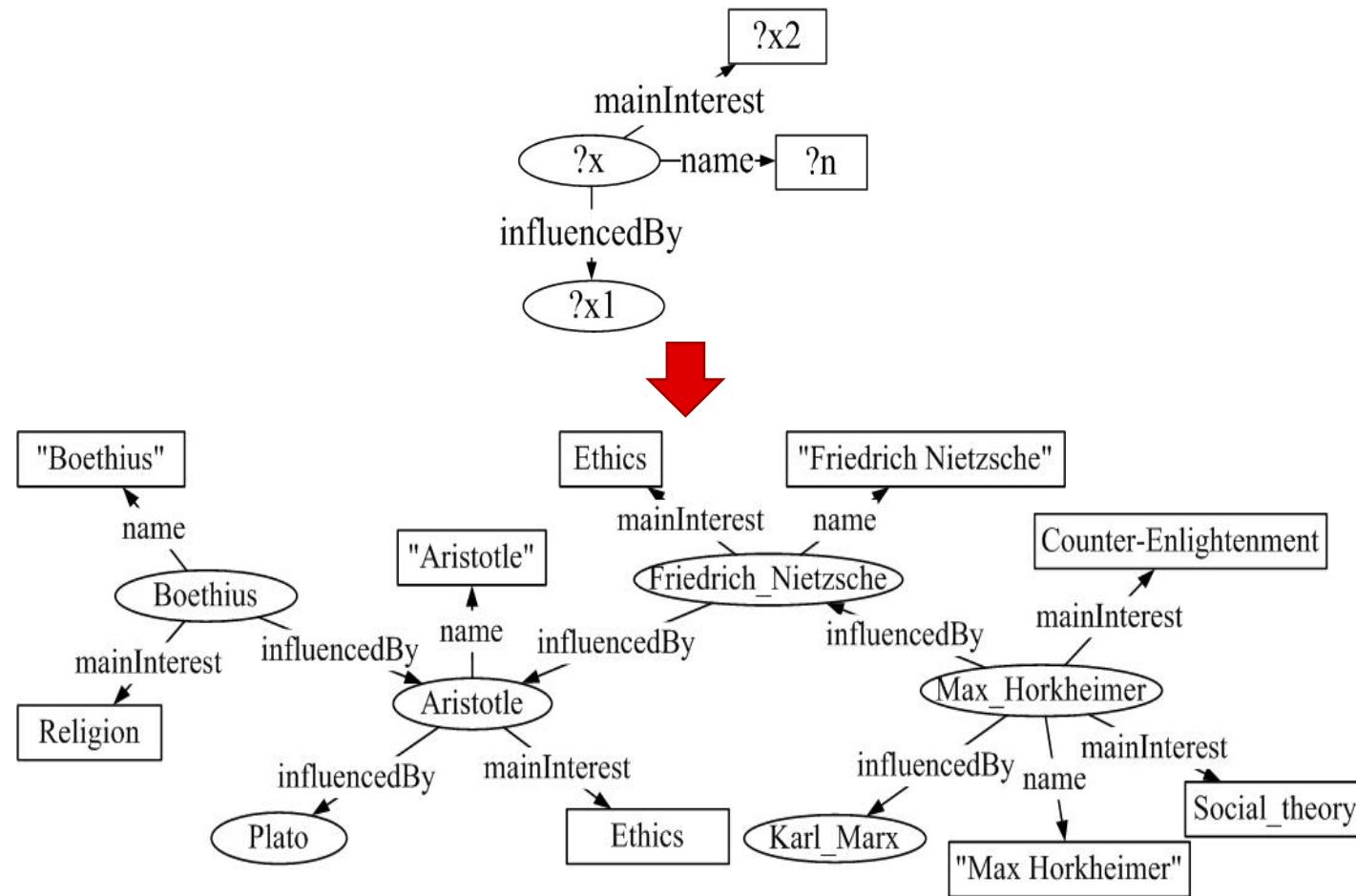
Algorithm 1: Frequent Access Pattern Selection Algorithm

Input: A set of frequent access patterns $P = \{p_1, p_2, \dots, p_x\}$

Output: A set $P' \subseteq P$ to generate fragments

```
1  $P' \leftarrow \emptyset;$ 
2  $TotalSize \leftarrow 0;$ 
3 for each  $p \in P$  and  $p$  has only one edge do
4    $P' \leftarrow P' \cup \{p\};$ 
5    $P \leftarrow P - \{p\};$ 
6    $TotalSize \leftarrow TotalSize + |E(\llbracket p \rrbracket_G)|;$ 
7  $P_1 \leftarrow \operatorname{argmax}\left\{\frac{\operatorname{Benefit}(\{p_i\}, Q)}{|E(\llbracket p_i \rrbracket_G)|} : p_i \in P, |E(\llbracket p_i \rrbracket_G)| + TotalSize \leq SC \wedge |E(p)| > 1\right\};$ 
8  $P_2 \leftarrow \emptyset;$ 
9  $TotalSize' \leftarrow 0;$ 
10 while  $TotalSize' \leq SC - TotalSize$  do
11   Find the frequent access pattern  $p' \in P - P'$  with the largest additional value of  $\frac{\operatorname{Benefit}(\{p'\} \cup P', Q) - \operatorname{Benefit}(P', Q)}{|E(\llbracket p' \rrbracket_G)|}$ ;
12    $P_2 \leftarrow P_2 \cup \{p'\};$ 
13    $P \leftarrow P - \{p'\};$ 
14    $TotalSize' \leftarrow TotalSize' + |E(\llbracket p' \rrbracket_G)|;$ 
15 if  $\operatorname{Benefit}(P' \cup P_1, Q) \geq \operatorname{Benefit}(P' \cup P_2, Q)$  then
16   Return  $P' \cup P_1;$ 
17 Return  $P' \cup P_2;$ 
```

□ 将所有与相同的频繁访问模式同构的匹配项放入同一个分区中。



- 我们将一个频繁访问模式的匹配项放入不同的分区中
- 我们扩展了简单谓词和最小项谓词的概念，将RDF图水平地分割。

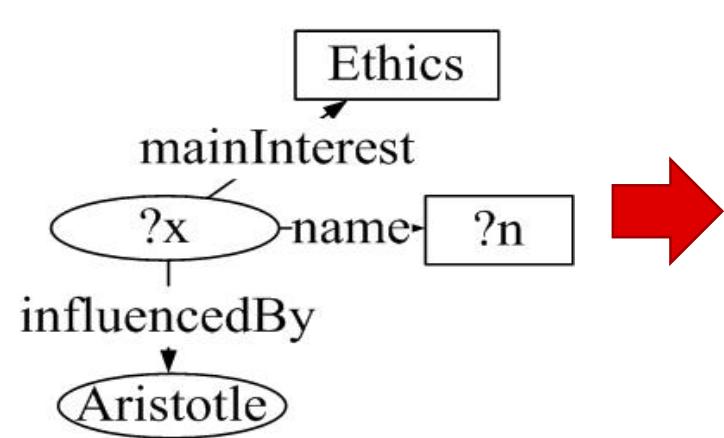
- 给定一个具有变量集 $\{var_1, var_2, \dots, var_n\}$ 的频繁访问模式 p ，在 Q 上定义的结构简单谓词 sp 具有以下形式

$$sp : p(var_i) \theta Value$$

- 其中 $\theta \in \{=, \neq\}$ ， $Value$ 是针对 var_i 的一个常量约束， var_i 是从包含 p 的查询 Q 中选择的。

- 给定一组针对频繁访问模式 p 的结构简单谓词 $SP = \{sp_1, sp_2, \dots, sp_y\}$, 结构最小项谓词 mp_i 是
- 其中 $sp_k * = sp_k$ 或者 $sp_k * = \neg sp_k$

示例结构简单谓词和最小项谓词



$sp_1 : p(?x1) = Aristotle;$

$sp_2 : p(?x1) \neq Aristotle;$

$sp_3 : p(?x2) = Ethics;$

$sp_4 : p(?x2) \neq Ethics.$



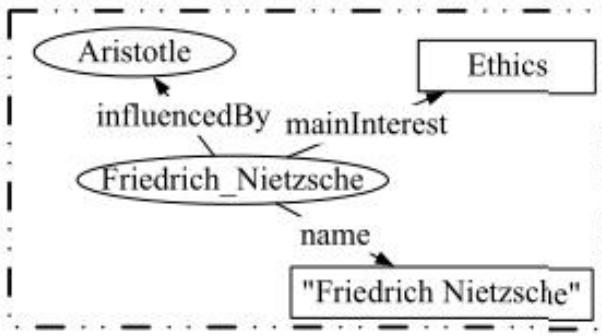
$mp_1 : p(?x0) = Aristotle \wedge p(?x1) = Ethics;$

$mp_2 : p(?x0) = Aristotle \wedge p(?x1) \neq Ethics;$

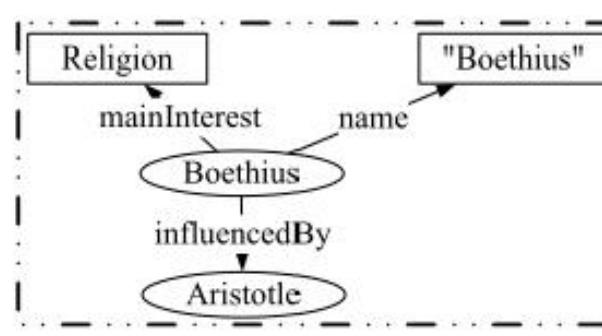
$mp_3 : p(?x0) \neq Aristotle \wedge p(?x1) = Ethics;$

$mp_4 : p(?x0) \neq Aristotle \wedge p(?x1) \neq Ethics.$

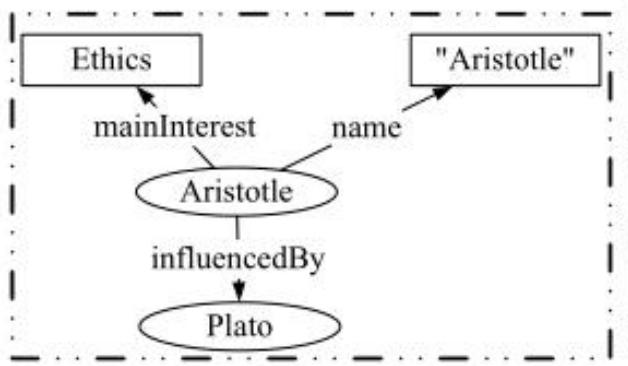
水平碎片化例子



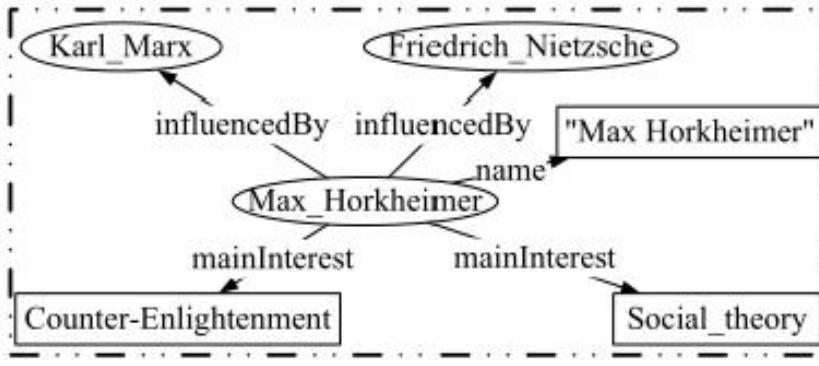
(a) Example Horizontal Fragment
Generated from mp_1



(b) Example Horizontal Fragment
Generated from mp_2



(c) Example Horizontal Fragment Generated from mp_3



(d) Example Horizontal Fragment Generated from mp_4

- 在我们的混合碎片化中，我们为垂直碎片化保留了某些频繁访问模式 (P^v)，但其他 (P^h) 是为水平碎片化设计的。

- 给定一个频繁模式 p , 让 $M(p)$ 表示从模式 p 派生的所有结构最小项谓词。
- 分类策略基于FAPs和相应的结构最小项谓词的访问频率, 将 $M(p)$ 分为两个子集:

$$M'(p) = \{mp \mid mp \in M(p) \wedge acc(mp) \geq minSup\}$$

$$M''(p) = \{mp \mid mp \in M(p) \wedge acc(mp) < minSup\}$$

- 给定一个选定的频繁访问模式 p , 我们有以下碎片化策略:
- 如果 $\sum_{mp \in M''(p)} acc(m|p) \geq minSup$, 我们根据模式 p 垂直分割RDF图,
即 $p \in P^v$;
- 如果 $\sum_{mp \in M''(p)} acc(m|p) < minSup$, 我们根据从模式 p 派生的频繁结构
最小项谓词 $M'(p)$ 水平分割RDF图, 即 $p \in P^h$ 。

- 如果两个分区经常被一起访问，则应将它们放置在一个站点中

□ 关于工作负载Q的两个分区F和F'之间的分区亲和力指标定义为如下

1. 给定由频繁访问模式p和p'生成的两个垂直分区F和F',

$$aff(F, F') = \sum_{k=1}^r use(Q_k, p) \times use(Q_k, p');$$

2. 给定由结构最小项谓词mp和mp'生成的两个水平分区F和F',

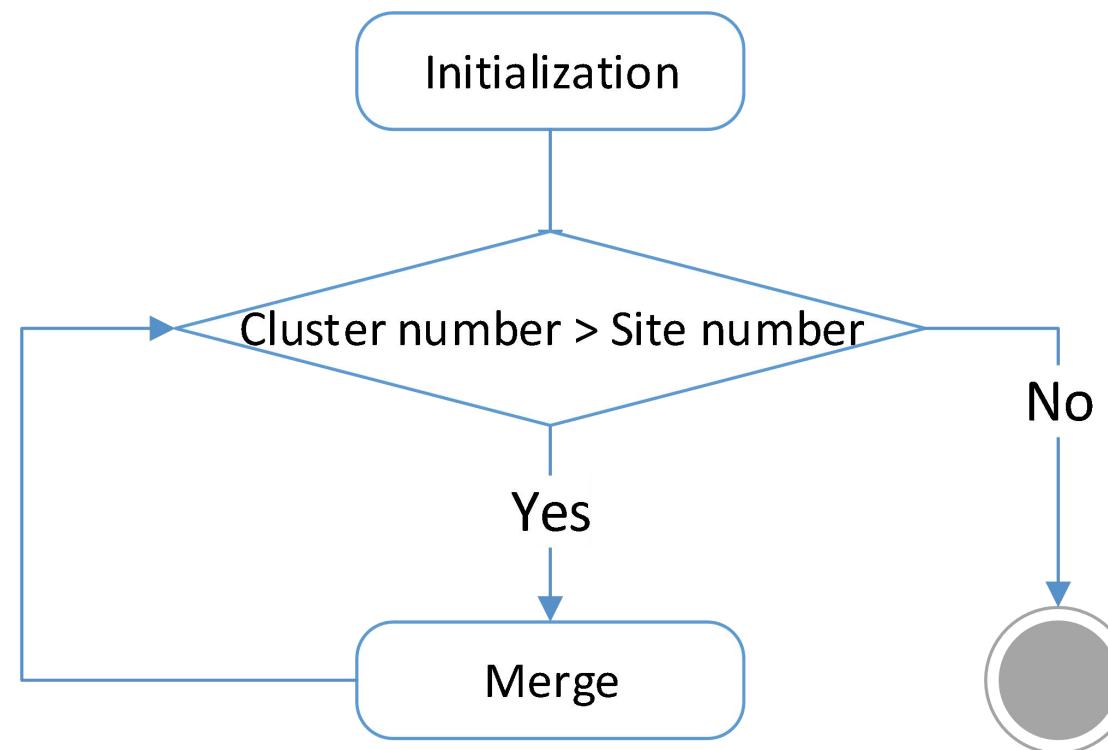
$$aff(F, F') = \sum_{k=1}^r use(Q_k, m\ p) \times use(Q_k, m\ p');$$

3. 给定由FAP_p和结构最小项谓词mp生成的两个混合分区F和F', 如果mp不是从p生成的, 那么

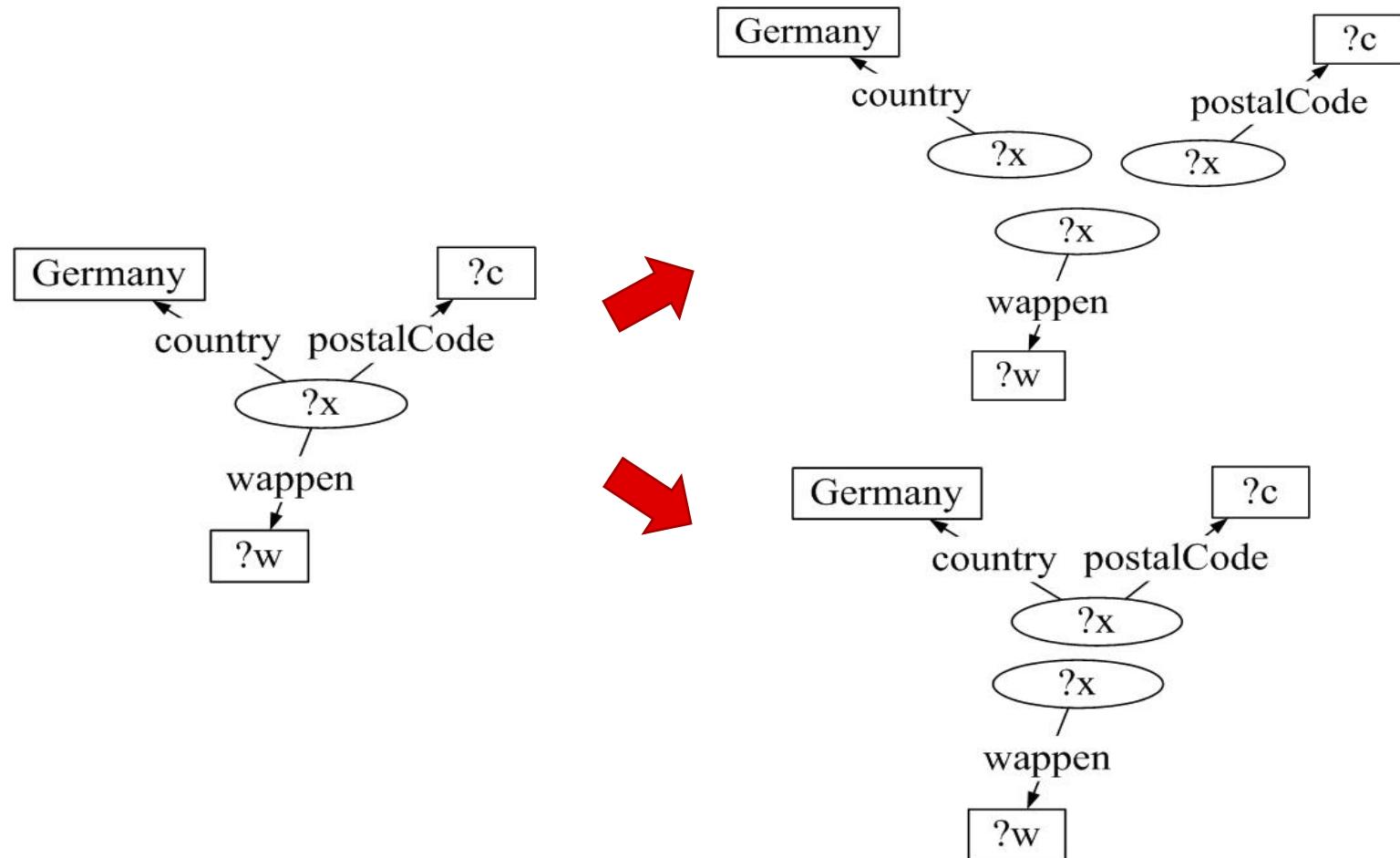
$$aff(F, F') = \sum_{k=1}^r use(Q_k, p) \times use(Q_k, m\ p);$$

否则, $aff(F, F')=0$

- 我们扩展了一个聚类算法PNN，以对所有分区进行聚类



□ 在查询评估过程中，我们首先会根据频繁访问的模式来分解查询。



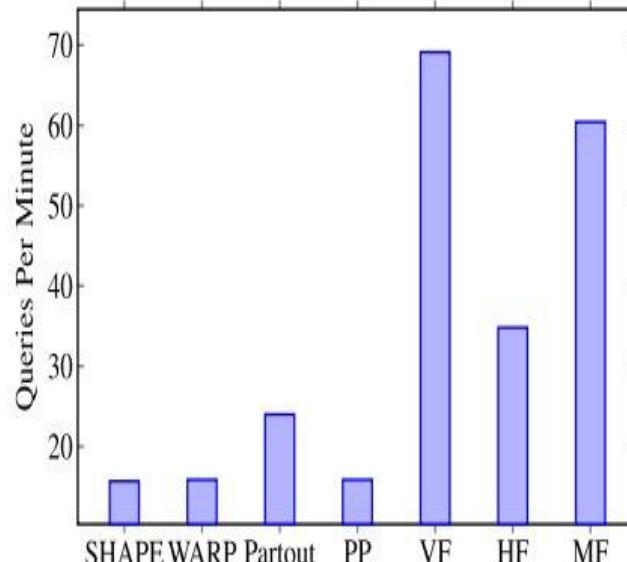
□ 查询分解的成本是

$$cost(D) = \prod_{q \in D} card(q_i)$$

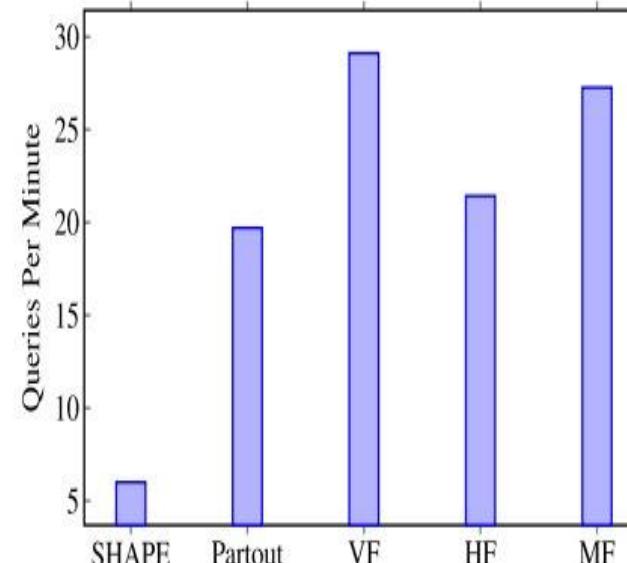
其中D是分解结果， $card(q_i)$ 是 q_i 的匹配数

- 在查询优化和执行方面，我们对System-R的算法进行了扩展，以找到将所有子查询结果联接起来的最优执行计划；
- 每个子查询都在相应的站点上并行执行；
- 然后根据最优执行计划将它们联接在一起。

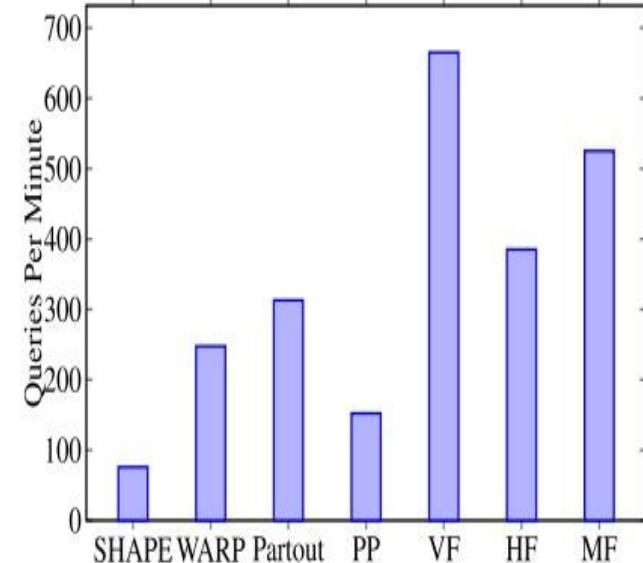
- **数据集**: 在对比测试中, 我们的系统在DBpedia (约3.3亿个三元组) 和 WatDiv (5000万到2.5亿个三元组)
- **工作负载**: 这两个数据集上运行了800万个DBpedia查询和2000个WatDiv 查询
- **竞争对手**: WARP和SHAPE
- **测试环境**: 在局域网内运行的10台Linux机器



(a) DBpedia

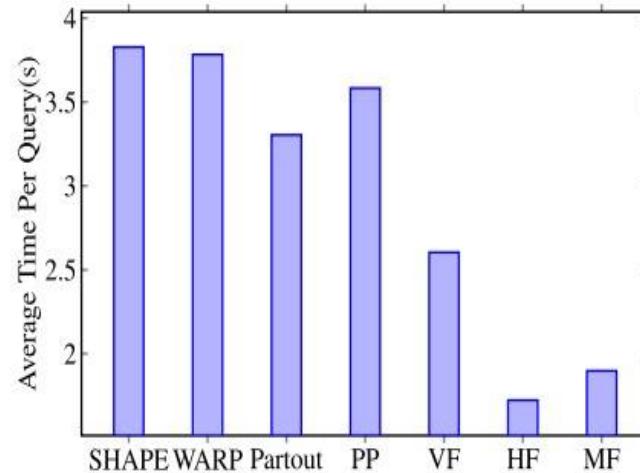


(b) LinkedGeoData

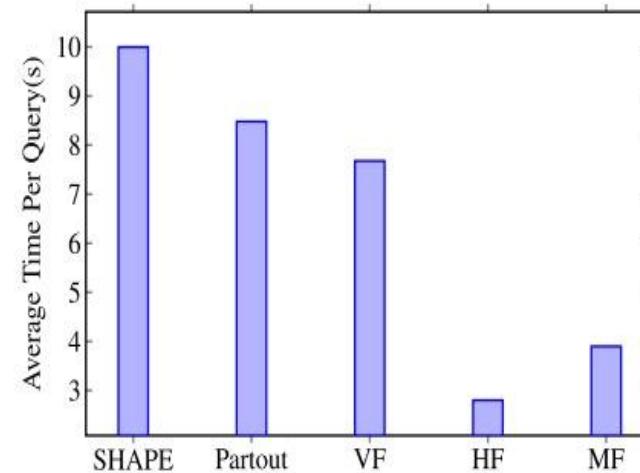


(c) WatDiv

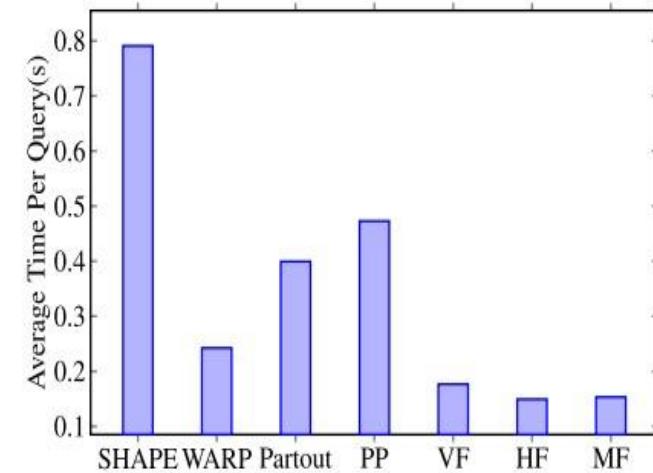
响应时间



(a) DBpedia



(b) LinkedGeoData



(c) WatDiv



湖南大学
HUNAN UNIVERSITY

第四部分

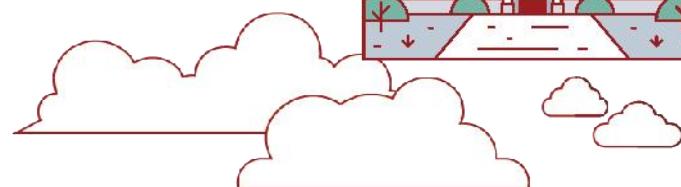
结论



- 我们引入了边不相交划分的分布式RDF系统，并将它们分为两类：与工作负载无关的方法和感知工作负载的方法。

谢谢观看

thank you for watching



- Alexander Schätzle, Martin Przyjaciel-Zablocki, Simon Skilevic, Georg Lausen. S2RDF: RDF Querying with SPARQL on Spark. Proc. VLDB Endow. 9(10): 804-815 (2016)
- Katja Hose, Ralf Schenkel. WARP: Workload-aware replication and partitioning for RDF. ICDE Workshops 2013: 1-6
- Peng Peng, Lei Zou, Lei Chen, Dongyan Zhao. Adaptive Distributed RDF Graph Fragmentation and Allocation based on Query Workload. IEEE Trans. Knowl. Data Eng. 31(4): 670-685 (2019)
- Peng Peng, Lei Zou, Lei Chen, Dongyan Zhao. Query Workload-based RDF Graph Fragmentation and Allocation. EDBT 2016: 377-388