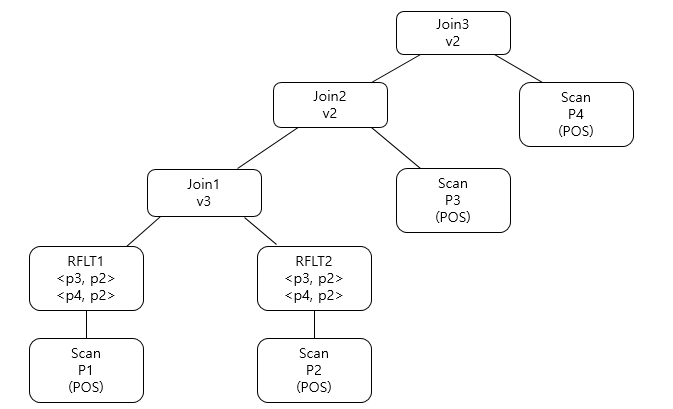
**1. RP-filter**

RP-filter는 불필요한 중간결과를 줄이기 위해 고안된 기술로 이 기술은 하나의 실행계획은 RFLT라는 필터링 연산자를 사용한다. Figure 5는 triple 필터링을 사용하여 하나의 실행계획을 보여준다. 여기서 두 개의 RFLT를 사용하고 있는데, RFLT 연산자는 입력된 트리플의 subject(Object)가 필요한 구조적조건을 만족하는지를 확인한다. 예를 들어 Figure 2에 있는 RFLT1 연산자는 들어온 두 개의 서술어(<p3, p2>, <p4, p2>)의 subjects를 찾는다. 이 조건을 만족하는 subject를 찾기 위해서 path-based index(RP-filter)를 사용한다. RP-filter는 vertex 리스트(들어오는 서술어 경로를 가진 ID)를 포함하고 있는데, 들어온 트리플과 vertex 리스트들은 결합함으로써 처리된다.



[ Figure 1 : Execution plan using triple filtering ]

1.1 Requirements of RP-Filter

RP-Filter 기술을 RDF 엔진에 적용하려면 RDF 엔진이 다음 요구 사항을 충족해야한다.

(1) URI는 정수 ID로 매핑되고 트리플은 세 개의 열, S, P 및 O가있는 트리플 테이블의 ID를 사용하여 저장된다.

(2) 실행 계획에는 SPARQL 쿼리의 각 트리플 패턴에 대해 하나의 스캔 연산자가 있으며, 이는 디스크에서 트리플 패턴과 일치하는 트리플을 읽는다.

(3) 스캔 연산자는 S 또는 O 열로 정렬 된 트리플을 읽는다.

RDF-3X를 비롯한 여러 RDF 엔진은 효율적인 쿼리 처리를 위해 위의 세 가지 조건을 충족하는 저장소 및 쿼리 처리 기술을 사용한다. 특히, 제 3 조건은 비교적 엄격하지만 조건은 여러 RDF 저장소에서 만족한다.(e.g. RDF-3X 및 SW-store.) 이는 트리플이 정렬 된 구체화로 인해 일치하는 트리플의 빠른 검색과 빠른 병합 조인의 사용과 같은 많은 효율성을 제공하기 때문이다. 따라서 RP-Filters는 RDF-3X 및 SW-store를 비롯한 다양한 RDF 엔진에서 채택 할 수 있다.

1.2 두 가지 Definition

RDF 데이터베이스의 RP-Filter는 노드 목록 집합이다. 술어 경로에 대한 노드 목록에는 술어 경로를 들어오는 술어 경로로 갖는 모든 노드 ID가 들어있다.

(1) Definition 1 (Node List of ppath N-List(ppath)).

술어 경로 ppath를 위한 노드리스트는 정렬된 노드(n)의 ID 리스트이다. 이는 DRW00002ab44497를 만족한다. N-List(ppath)는 하나의 술어 경로 ppath의 리스트를 나타낸다. 하나의 술어 경로를 위한 노드리스트를 읽으면 술어경로로 표현되는 경로패턴에 의해 도달되는 모든 ID들을 쉽게 얻을 수 있다.

(2) Definition 2 (RP-filter of RDF database D with the maximum length MaxL).

RDF 데이터베이스 D와 최대길이인 MaxL가 주어지면 D의 RP-filter 모든 ppath, N-List(ppath)의 집합이다. 모든 ppath는 D안에 존재하며, 그것의 길이는 MaxL보다 작거나 같다.

|  |
| --- |
| DRW00002ab44499 |

|  |
| --- |
| EMB00002ab44491 |

[figure 2 : RP-filter (MaxL = 3)]

하나의 술어 경로 ppath는 D안에 존재하고, MaxL은 RP-filter의 사이즈를 제한한다.

MaxL이 증가함에 따라, 술어경로의 수는 증가한다.

Figure 6는 술어 경로와 노드리스트들을 보여준다. RP-filter(D,3)안에는 11개의 노드리스트가 존재하고, 노드 ID를 기준으로 정렬된 각각의 노드리스트를 볼 수 있다.

1.3 Storage Model of RP-Filter

각 노드 목록은 노드 ID별로 정렬된 상태로 디스크에 저장되므로 노드 목록을 트리플 필터링을 처리하는 동안 정렬된 순서로 읽는다. RDF-3X에서 사용 된 델타 기반 블록 압축 기술을 사용하여 RP 필터의 크기 오버 헤드와 노드 목록을 읽는데 필요한 디스크 I/O 오버헤드를 줄인다. 이 방법에서는 인접한 두 노드 ID의 차이가 저장된다. 델타 크기에 따라 델타 쓰기 위한 바이트 크기가 결정된다. 노드 ID가 정렬되기 때문에 이 압축 방법을 사용할 수 있다.

|  |
| --- |
| EMB00002ab44492 |

[Figure 3 : RP-Trie for RP-Filter(D, 3)]

술어 경로를 노드 목록에서 효율적으로 검색하기 위해 RP-Trie라는 트리에서 조건부 경로를 구성한다. RP-Trie는 RP-Filter의 모든 술어 경로로 작성된 트리를 말한다.

RP-Trie의 레벨 DRW00002ab4449b에 있는 각 노드에는 연관된 길이 DRW00002ab4449d술어 경로에 대한 노드 목록에 대한 포인터가 있다. Figure 7는 Figure 6의 RP-Filter (D, 3)에 대한 RP-Trie를 보여준다. 술어 경로를 사용하여 RP-Trie를 탐색함으로써 노드 목록의 디스크에서 위치를 찾을 수 있다. 길이가 MaxL 이하인 술어 경로에 대한 노드가 없다면, RDF 데이터베이스에서 술어 경로와 일치하는 경로가 없다고 결론을 내릴 수 있다.

RP-Trie는 최악의 공간 복잡성을 가지며, MaxL에 기하급수적으로 증가 할 수 있다. 실제 데이터 세트 및 작은 M axL 값의 경우 RP-Trie의 크기는 상대적으로 작으며 RP-Trie는 주 메모리에 상주 할 수 있다. RP-Filter의 구조는 정보 검색 영역에서 널리 사용되는 역 인덱스 구조와 유사하다. RP-Filter의 각 술어 경로는 거꾸로 된 색인의 어휘집으로 간주 될 수 있지만 노드 목록은 게시 목록과 거의 같다. 거꾸로 된 색인의 게시 목록이 문서 ID를 정렬 된 순서로 유지하는 것처럼 노드 목록은 노드 ID를 정렬 된 순서로 유지한다. 술어 경로는 술어 경로에 대한 노드 목록을 찾고 관련 노드 ID를 빠르게 찾는 데 도움이 되도록 RP-Trie로 구성된다.

1.4 RP-Filter Size

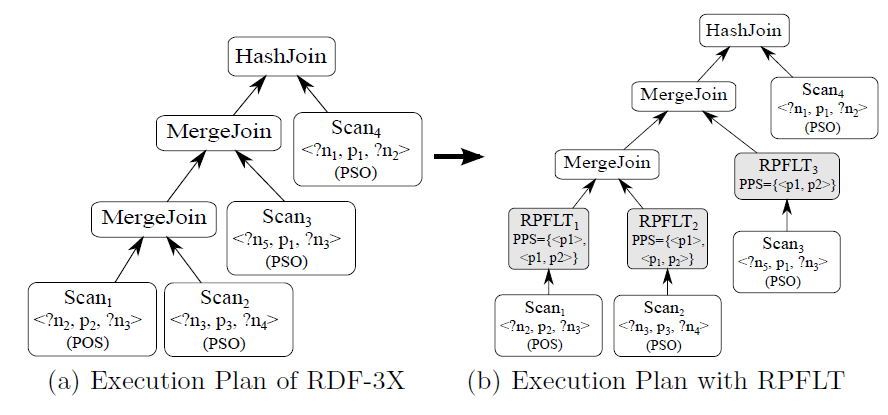
두 개의 데이터 세트에 대해 MaxL = 3 인 RP 필터를 만든다. 표 3은 N-List의 수와 RP-Filters의 전체 크기를 보여줍니다. 보는 것과 같이 두 개의 데이터 세트에 대한 RP-filter의 크기는 입력 데이터 세트 크기보다 훨씬 작다. YAGO2의 데이터 크기가 LUBM의 크기보다 작더라도 YAGO2에 대한 N-List의 수는 LUBM보다 많다는 걸 알 수 있다. 이것은 YAGO2가 LUBM보다 더 많은 술어를 가지고 있기 때문에 YAGO2에서 보다 다른 술어 경로가 존재하기 때문이다.

[Table 1 : RP-Filter Size(MaxL=3)]

|  |  |  |  |
| --- | --- | --- | --- |
|  | N-Lists | total size | avg. length |
| YAGO2 | 39,635 | 836MB | 16,305 |
| LUBM | 122 | 1.3GB | 2,571,504 |

2 Query Evaluation Using RP-Filter

2.1 RPFLT Operator



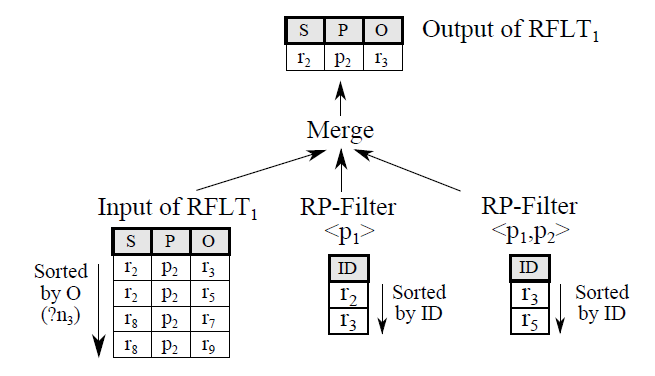
[Figure 4] Application of RPFLT Operators

RP-Filter는 실행 계획에서 스캔 연산자의 triples를 필터링 하는 데 사용됩니다. RP-Filter를 사용하기 위해 조회 컴파일러는 실행 계획에 RPFLT라는 연산자를 추가한다. RPFLT연산자는 자식 스캔 연산자를 통해 자식 스캔 연산자를 가져오고 RP-Filter를 통과하는 트리플 파일만 출력합니다. 실행 계획의 상위 연산자로서 RPFLT연산자가 실행 계획에 추가 교환한다.

RPFLT의 예측 가능한 경로 집합. RPFLT연산자는 쿼리 컴파일러가 할당한 PPS(PredicatePathSet)라는 일련의 술어 경로를 가지고 있다. RPFLT의 PPSPPS에 어떤 선행 경로를 포함할 수 있는지를 설명하기 위해, 우리는 다음과 같이 검색 연산자라 불리는 스캔 연산자의 속성을 정의한다. 검사 연산자의 결과 triple은 S또는 O열에서 명령합니다(술어 변수와 술어를 고려하지 않으므로 명령 열이 아님). 결과 triples가 스캔 연산자의 Sortkey열 정렬을 정렬하는 열을 호출합니다. Sortkey열에는 매핑 된 조회 그래프에 해당 노드가 있습니다. 또한 Sortkey노드를 사용하여 쿼리 차트에 있는 Sortkey열에 해당하는 노드를 나타냅니다. Scani.kertkey는 컨텍스트에 따라 Shankey열 또는 Shani의 Sortkey노드를 나타내는 데 사용됩니다. 그림. 5(a)그림의 조회 그래프에 대한 RD-f9의 예제 실행 계획을 보여 줍니다. 2(b) 각 스캔 연산자의 마지막 항목은 사용할 인덱스 유형입니다. 예를 들어 Scance1은 POS색인을 스캔하고 스캔한 triples는(P, O, S)순으로 주문합니다. 모든 트립은 동일한 술어 값인 'p2'를 가지므로, 실제로는 O열로 정렬됩니다. 이와 같은 방법으로 Scant2의 결과는 S열에 의해 주문된다. 따라서 Scank1의 Sortkey열은 'O'이고 스캔키의 Sortkey열은 'S'이다. 그리고 Scank1과 Scane2의 Sortkey노드는 모두 노드입니다. 그림 3에서 볼 수 있습니다. 2(a), 즉, Scank1.sortkey=Scant2.kertkey=?3. Scani에 대한 RPFLT의 PPS는 Infrath의 하위 집합이어야 한다(Scani. 행운을 빕니다 예를 들어, scsi1.surtkey=?1은 Scant1.3.3및 p1, p2, p2, p2에 있는 RPFLT의 PPSPPS에 포함될 수 있습니다. 그림. 5(b)2개의 RPT연산자를 사용하는 실행 계획을 보여 준다. RPFLT연산자의 마지막 항목은 PPS에 있는 술어 경로를 열거한다. Scank1과 Scane2의 RPFLT연산자는 동일한 PPS를 가지고 있다. 두 스캔 연산자의 정렬이 동일하기 때문이다. PPS는{\_p1, p2\_,\_p\_\_p\_}이지만, Infrath(?3)는{\_p1, p2\_p\_\_p\_\_p\_}입니다. PPS에 세개의 서술 경로가 포함된 이유는 5.2절에 설명되어 있다. Scankey3연산자에는 소프트 키 노드가 들어오지 않는 이유로 인해 RPFLT연산자가 없다.

2.1 Filtering Process of RPFLT. An RPFLT

연산자는 PPS의 서술 경로에 대한 모든 F-IFSP에 대한 값을 포함하는 트립 막대를 출력합니다. 그림. 그림 6에서는 그림 1의 스캔 프로세스를 보여 줍니다. 5(b) 필터링 프로세스는 입력 트리를 모든 할당된 PEM으로 병합합니다. 이 예에서는 스캔이 네개의 트리플을 출력하지만, 세개 중 세개는 RPFLE1에 의해 필터링 된다. RPFLFLE1은 N-list(N1, p2, p2, N-list)와 N-list(p1-list)의 'r3'에 해당하는 3개의 트리플을 출력한다.



[Figure 5] Filtering in RPFLT Operator

쿼리 그래프의 ?n3 노드(Scank.cortkey)는 Inpeath에 있는 두개의 설명 경로를 가지고 있습니다. 따라서 일치하는 데이터 노드가 있다. 3개의 들어오는 기술 경로도 있어야 합니다. 두개의 F-IFST가 교차하는 교차점은 두개의 들어오는 기술 경로가 모두 있는 일치하는 데이터 노드를 제공합니다. 필터링 된 세개의 삼둥이가 필터링 되지 않은 경우, Mergezoin과 HashJoin은 다음 번 조회 결과에 기여하지 않고 전체 Join 작업을 수행하고 전반적인 쿼리 처리를 늦춥니다. 이러한 방식으로, 우리는 최종 결과에 포함되지 않는다면 가능한 한 빨리 세 단계를 걸러 내려고 노력한다. 필터링 프로세스에는 노드 목록을 읽고 입력 트리플과 병합하는 것이 포함됩니다. 동일한 ordering을 공유하기 때문에 노드 목록과 입력 트리를 병합하여 입력 트리를 필터링 할 수 있습니다. 또한 E-IFST는 대개 작은 길이입니다. 따라서, L-IFSTER의 판독 값과 병합은 오버 헤드가 거의 발생하지 않으며, RPFLT연산자는 매우 효율적이고 경량화됩니다.