Efficient distributed query processing for autonomous RDF databases

Theivapulendra E.

Overview

1. Introduction and previous work
2. Proposed approach
3. Query optimization
4. Query Execution
5. Avalanche
6. Discussion

Section 1

INTRODUCTION AND PREVIOUS WORKS

Autonomous RDF Databases

- Drug development database
- Clinical /Medical records
- **Restrictions**
  - Access permission on demand and in conformance with local authorization models
  - Not allowed to store data in central environment

Example of Distributed Query Processing

- the titles of articles by employees of organizations that have projects in the area "RDF"
  - title {T};
  - author {W} affiliation {O}
  - carriesOut {P} topic {'RDF'}

Available data sources

S_1: publication data base about articles, titles, authors and their affiliations
S_2: project data base about industrial projects, topics, and organizations

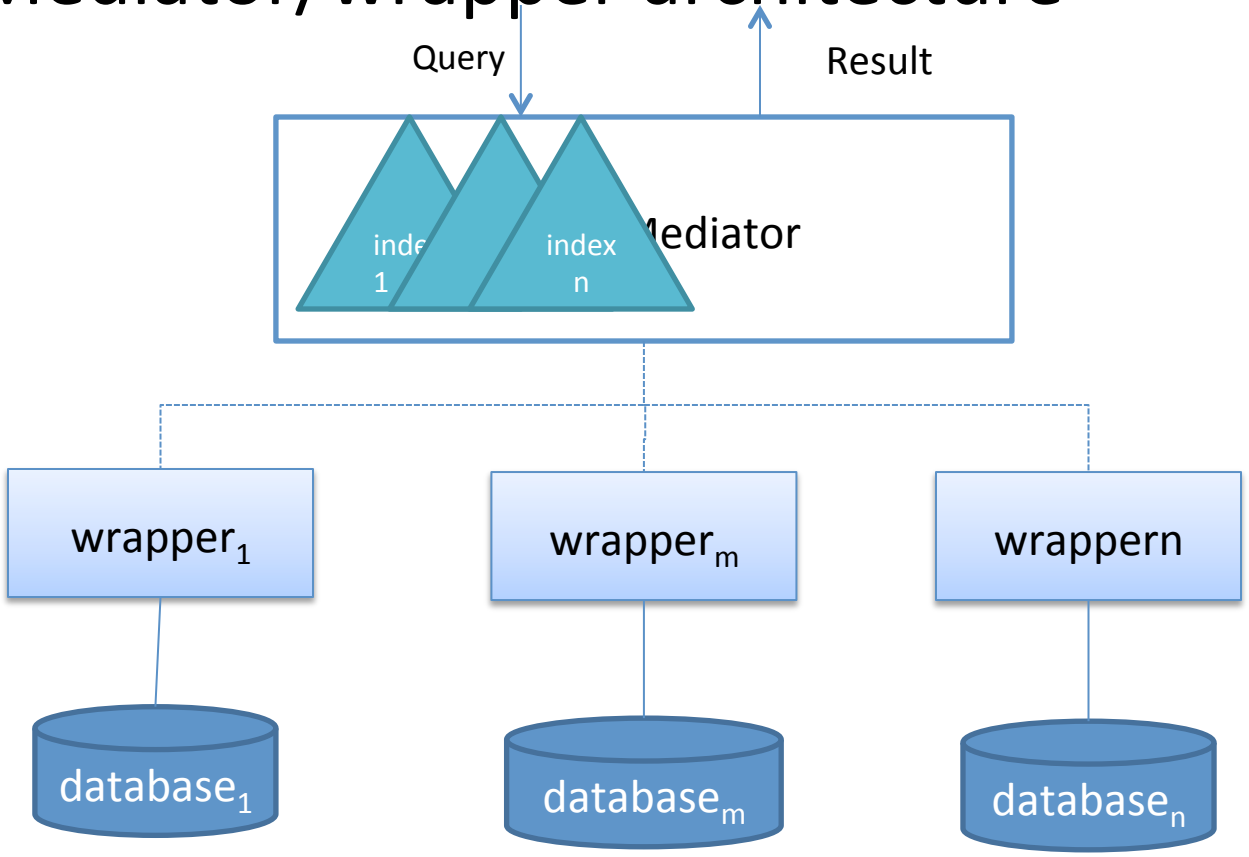
So fragment of query

title {T}; author {W} affiliation {O} >>>>> S_1
{O} carriesOut {P} topic {'RDF'} >>>>>> S_2
And join the results

How are we going to direct the part of the query to correct data source????

Previous works

- Mediator/wrapper architecture

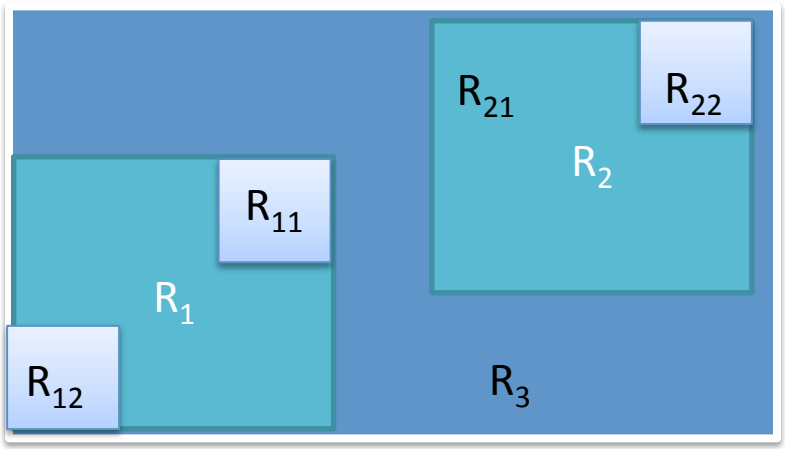


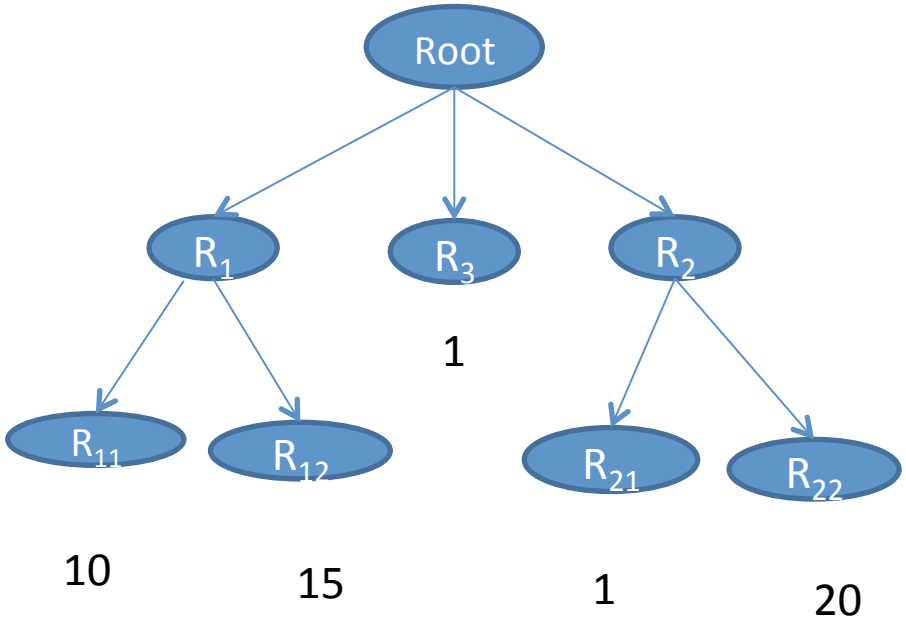
Limitations of mediator pattern

- Data source selection is based on predicates in triple pattern.
- Predicates must be bound because it is the basis for the pattern definition.
  - Handles on the patterns with predicate contains URI

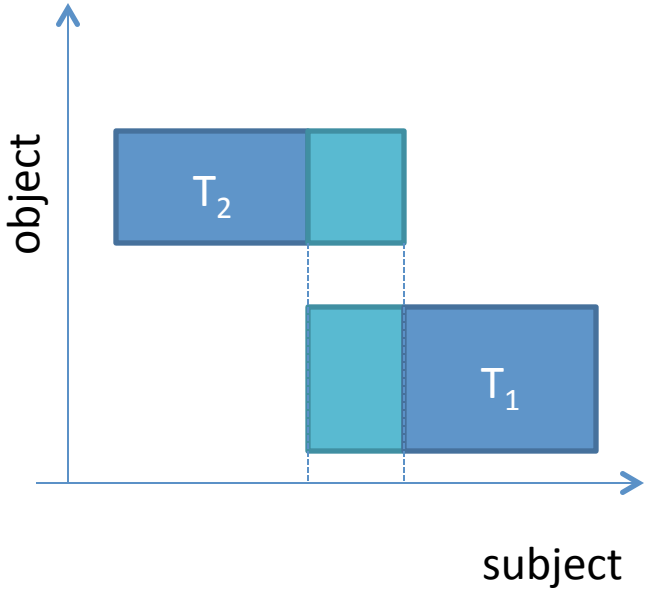
Data Summaries

- Based on schema and instance level index
- Triple patterns are hashed and mapped to a histogram
- Q-Tree index based on histogram





Join operation on data summaries



Dependencies between triples T1 and T2 decided by the overlapping Minimum Bounding Boxes (MBB's)

Limitations of data summaries

- Due to main memory requirements and running time, Data summaries can handle only simple queries.
- Due to hashing , type information is lost
- Joining overlapping regions is inappropriate for large triple set because Q-Tree allows a limited number of leafs.

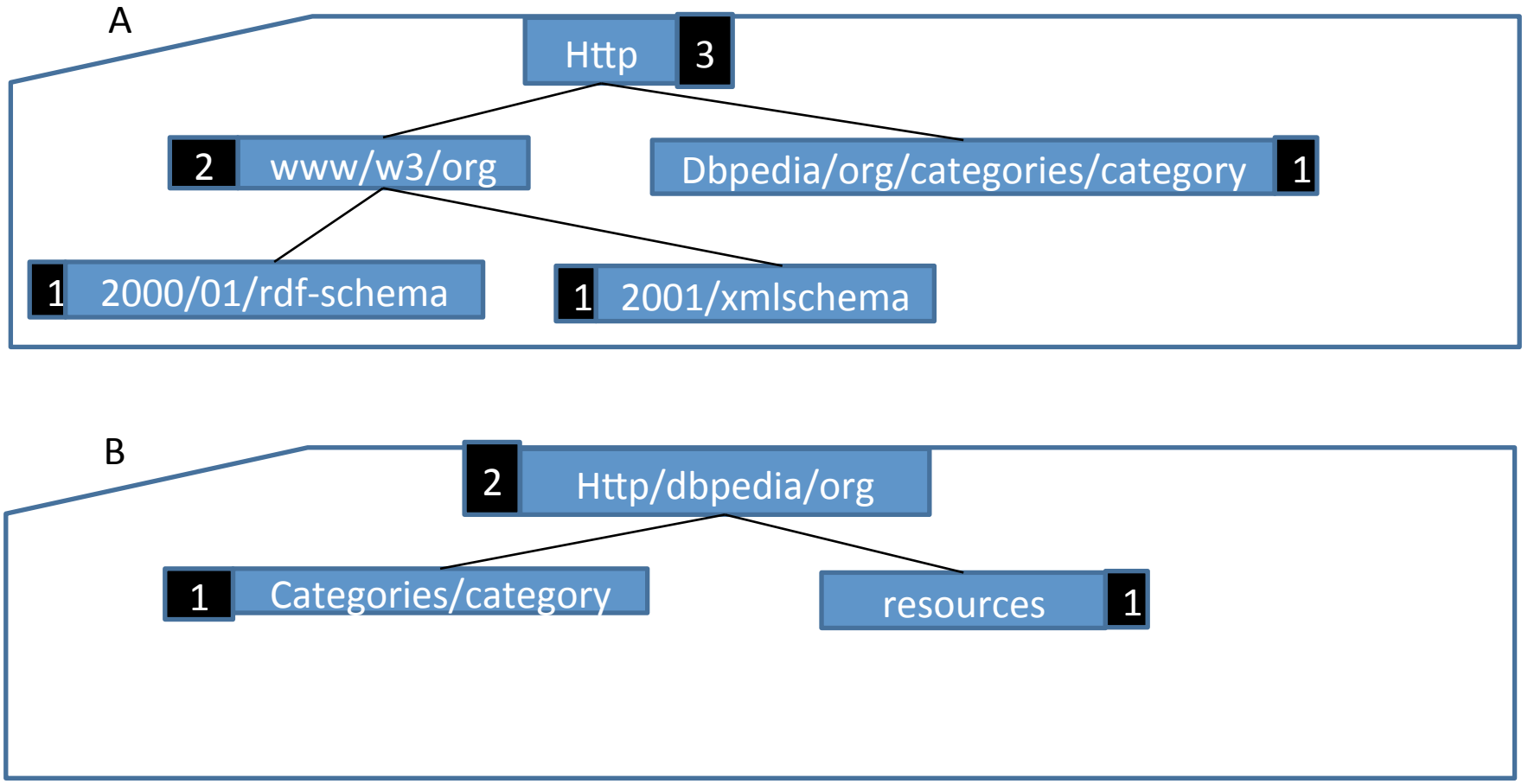
Section 2

PROPOSED APPROACH

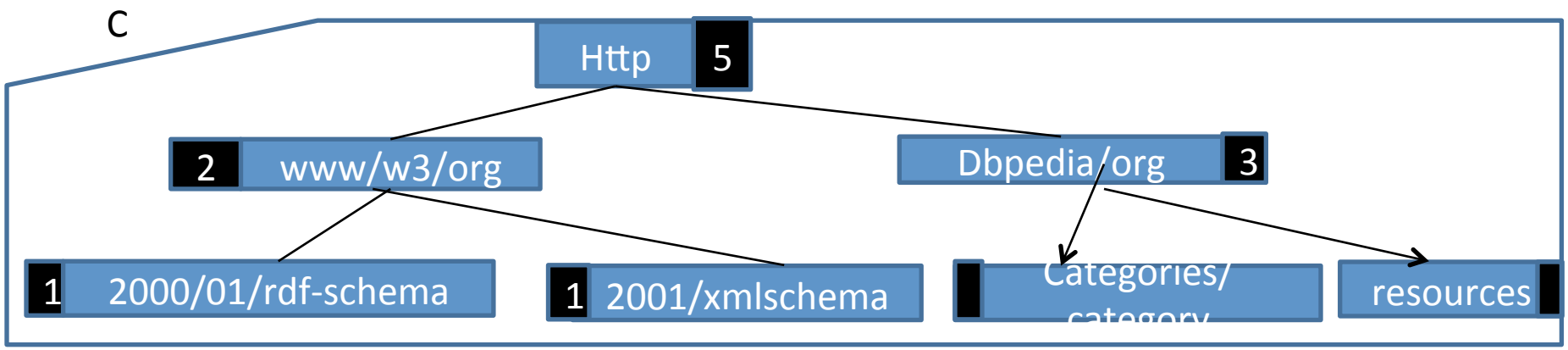
Extending Indexing of Data Summaries

- Preserving Type information
  - 2 byte integers are added to literals and URI
  - When hashing triples these values are kept as is.

- How URIs' typed?
  - Values are assigned based on prefixes of URI
  - Determine the common set of prefixes in data sources
  - Normalize and split them into a list of path components
  - Construct the global view to map data sources to same type information


A
Http
3
2
www/w3/org
Dbpedia/org/categories/category
1
1
2000/01/rdf-schema
1
2001/xmlschema
B
2
Http/dbpedia/org
1
Categories/category
resources
1

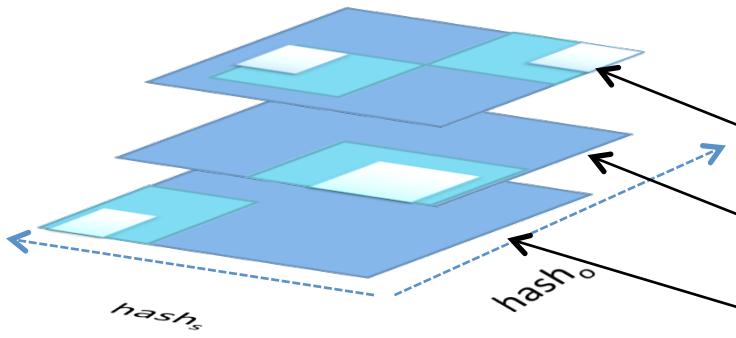
Global prefix tree



Vertical partitioning of indexes

- Group triples that share the same predicate into a common partition
  - Reduce the amount of redundant data
  - Preserve predicates' hash value
  - Handle type information efficiently

Index organization with partition



Id
DB_A
DB_B
DB_C

$type_s$	$type_p$	$type_o$	$hash_p$
13	39	32	324321007
25	02	44	175307839
04	12	19	921213243

Partition-Trees: Extension of Q-trees

- Leaf nodes contains additional information: 2 LSB of hash(t_s) and hash(t_o)
- N.lsb={(hash(t_s) mod 2^{16}, hash(t_o) mod 2^{16})}
- It helps to improve the accuracy in locating the sources by reducing the overlapping regions.
- During indexing the PARTrees are generated locally by applying the Sort-Tile-Recursive bulk-loading algorithm and transmitted to the mediator on a per partition basis

SUMMARY

- Partition Table
  - Information about hash value of predicate and type information of subject, object and predicate
  - One table per each data source partition by predicates' hash value
- PARTrees
  - Information about hash value of subject and object
  - Information of MBB's of regions

Section 3

QUERY OPTIMIZATION

Initial Plan Generation

- Selecting possible sources for each triple pattern in the query
- For a triple $t=(t_s,t_p,t_o)$ check the partition table
  - t_s is a variable or $type(t_s)=p.type_s$
  - t_p is a variable or $type(t_p)=p.type_p$ and $hash(t_p)=p.hash_p$
  - t_o is a variable or $type(t_o)= p.type_o$

Get the range or point queries from partition table

- Execute against PARTrees referenced by partitions
  - Traverse from root to leaf which consists of valid nodes n
    - A leaf node n_1={min_s,max_s,min_o,max_o,lsb}
    - N.min_d ≤ hash(t_d) ≤ n.max_d where d ϵ {s,o}
    - LSBs' match the hash values of triple
  - In a valid leaf node following are true
    - t_s and t_o are variables or
    - t_o is a variable and (hash (t_s) mod 2^{16},y) ϵ n_1.lsb, or
    - t_s is a variable and (x, hash(t_o) mod 2^{16}) ϵ n_1.lsb, or
    - (hash (t_s) mod 2^{16} ,hash (t_o) mod 2^{16}) ϵ n_1.lsb

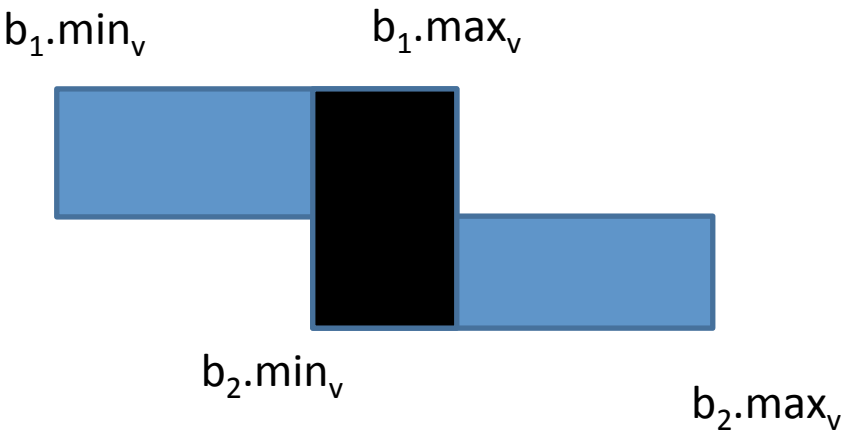
Plan Simplification

- Binding results from different sources
  - By evaluating remote sources , approximate variable bindings *b* can be retrieved with type information and boundary hash value
  - From the partition tree node's MBB's
    - Lower and upper bound for variable hash value b_{max}, b_{min}
    - $b.bitset_v$ from the node's LSBs encoded in 2^{16}
  - From partition table
    - $b.type_v$ from the type information

Extension on Join Operation of data summaries

- Data types and bitsets of AVB's considered
  - Two AVB's b_1 and b_2 can be joined , if the following true
    - $b_1.min_v \leq b_2.max_v$ and $b_1.max_v \geq b_2.min_v$ (overlapping)
    - $b_1.type_v = b_2.type_v$ (types are equal)
    - $(b_1.bitset_v$ AND $b_2.bitset_v) \neq 0$ (at least one equal bit is active in the bitsets)

Properties of resulting AVB



- $b_r.min_v = max(b_1.min_v, b_2.min_v)$
- $b_r.max_v = min\ (b_1.max_v\ , b_2.max_v)$
- $b_r.bitset_v = b_1.bitset_v$ AND $b_2.bitset_v$
- $b_r.type_v = b_1.type_v$

Postprocessing

- Transform operation tree and derive sub queries
- Prune redundant dispatch operation that has no contribution to result

Query Optimization Example

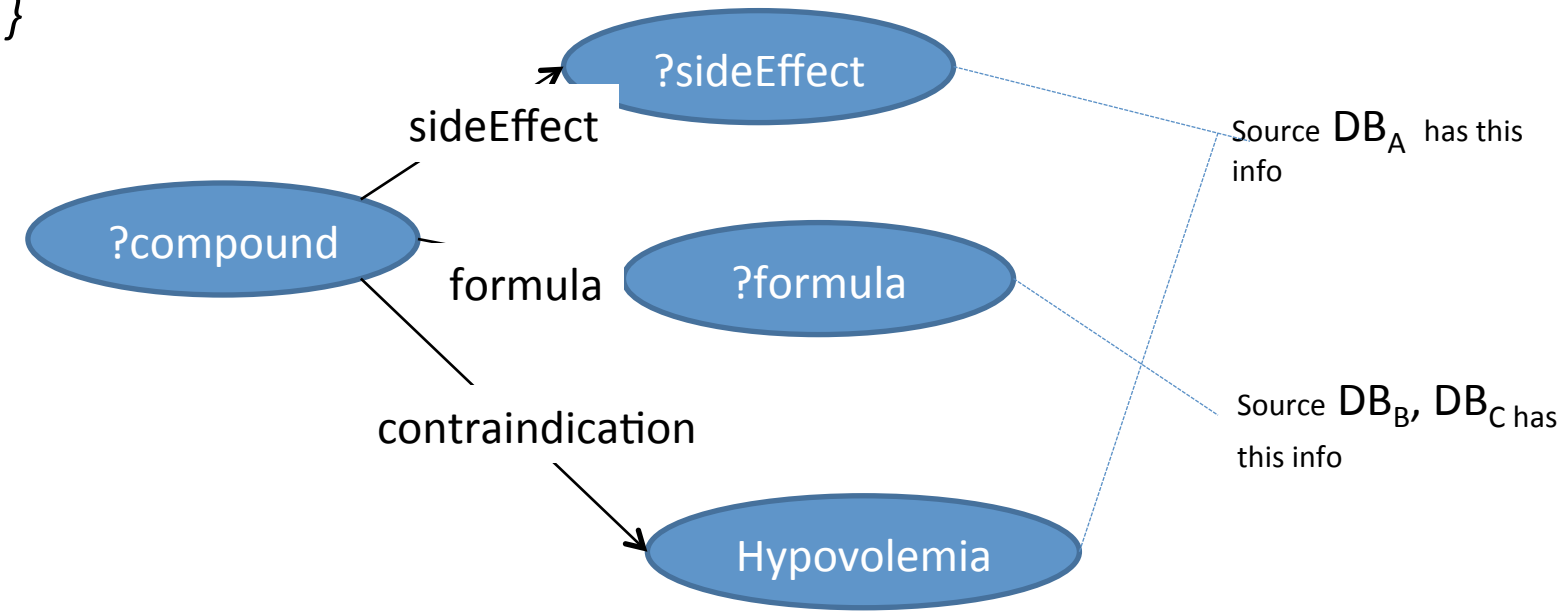
- *Select ?compound ?formula ?sideeffect WHERE {*

  ?compound contraindication Hypovolemia,

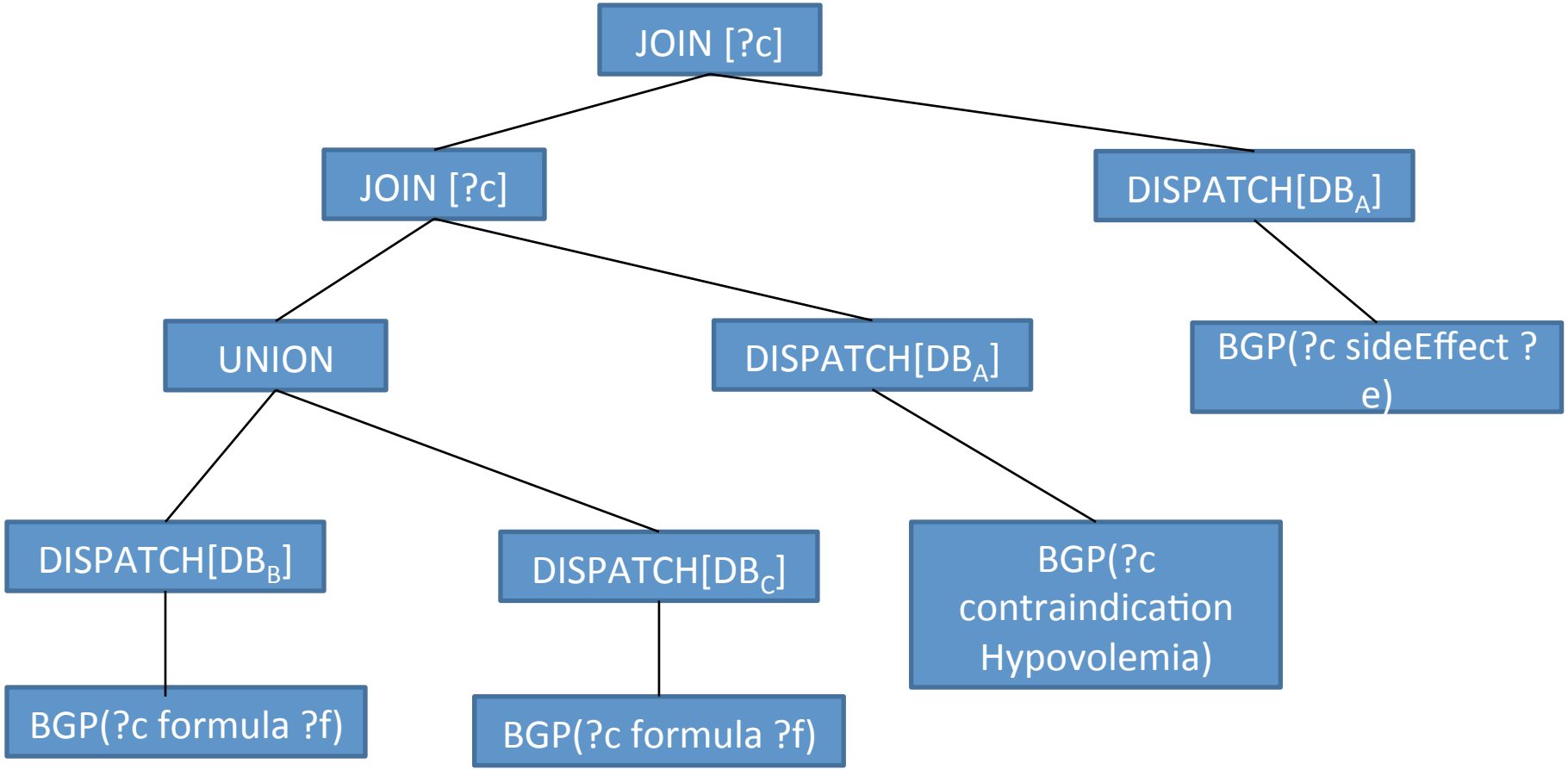
  ?compound formula ?formula,

  ?compound sideEffect ?sideEffect

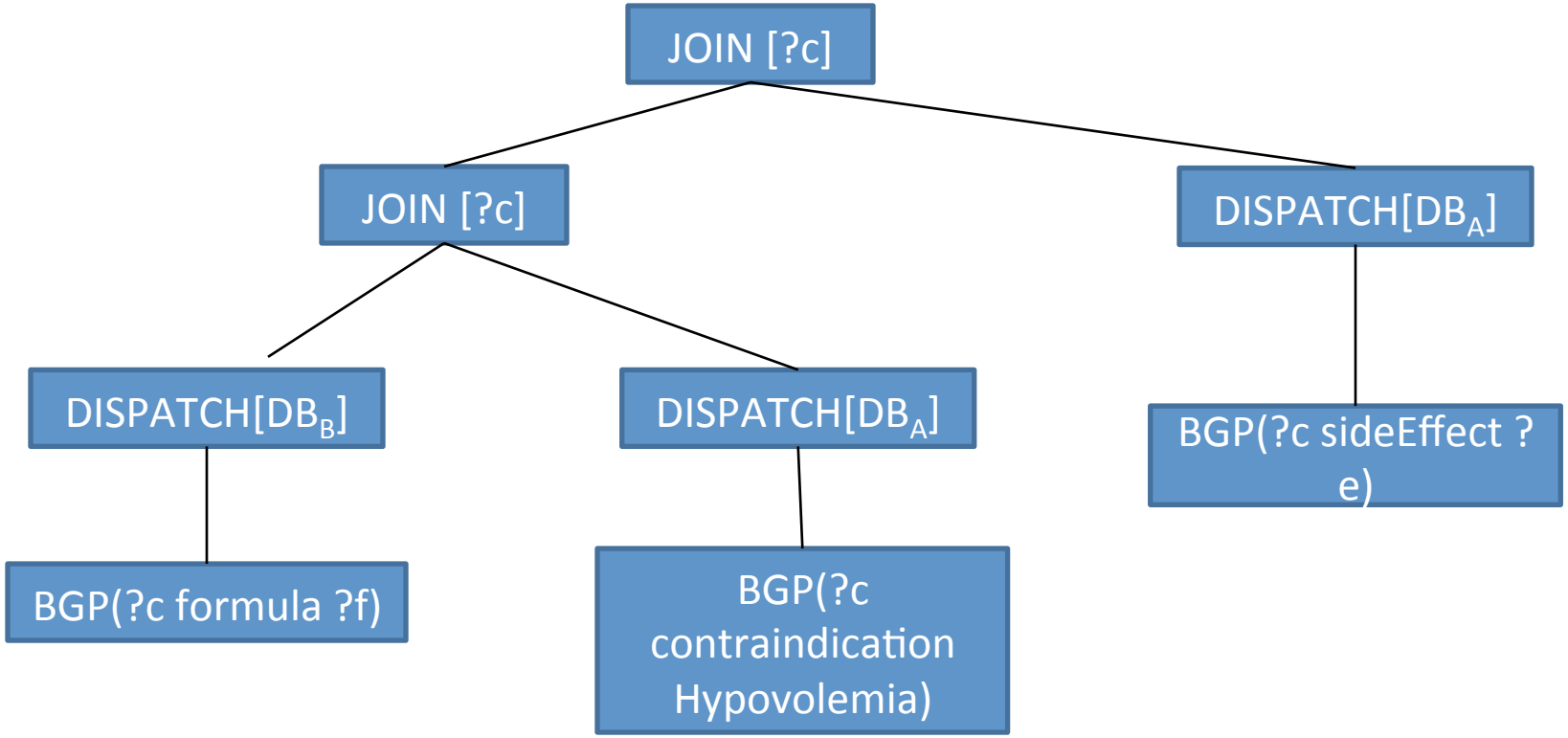
  }



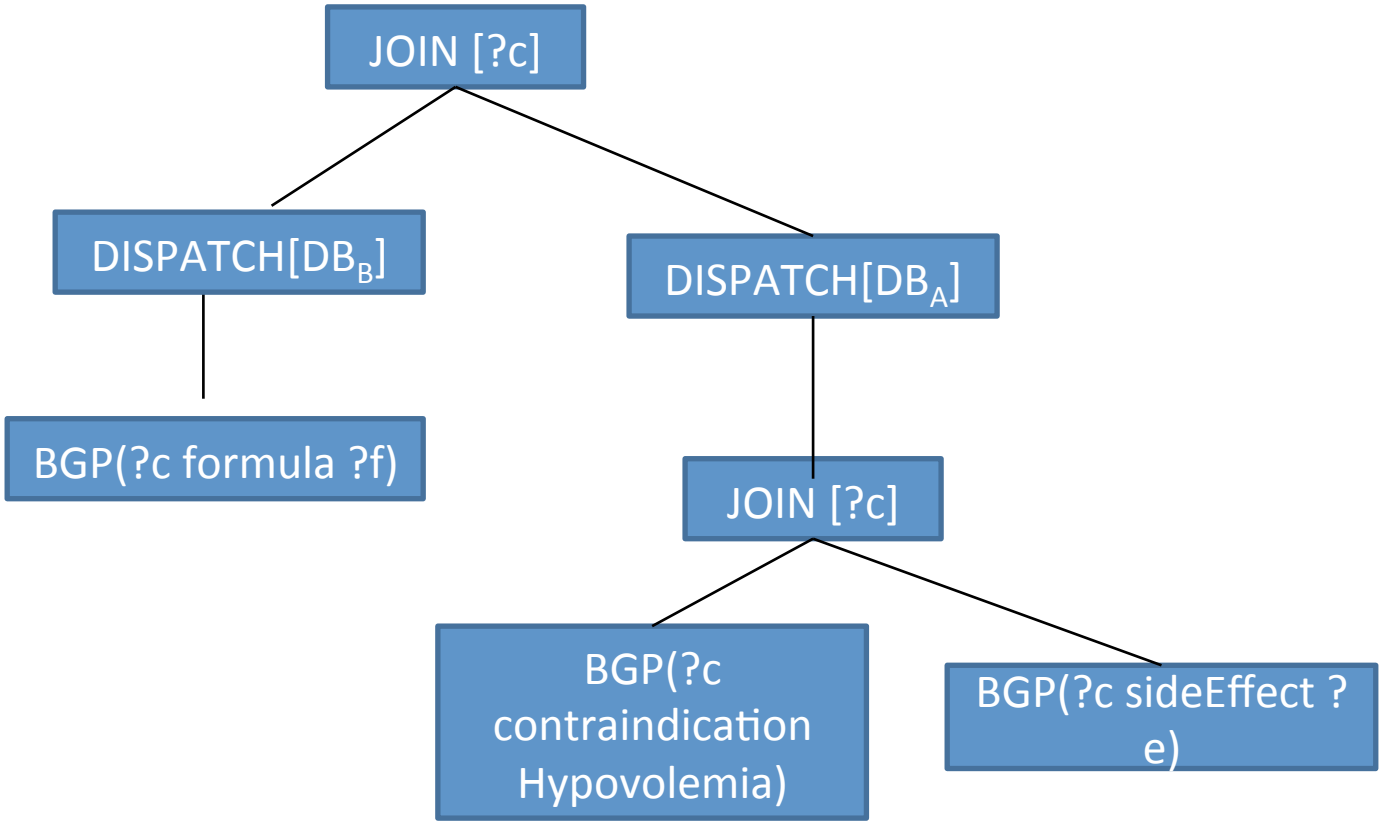
– Decide data sources for triples
– Decide join variable binding

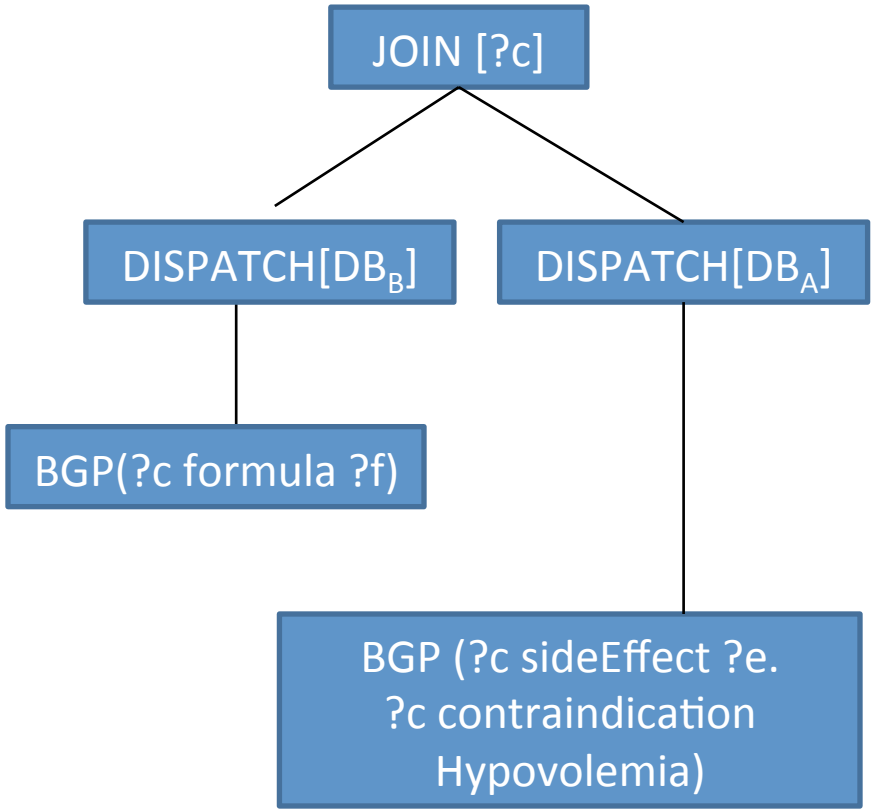


– Prune redundant dispatch-operators



– Reorganize operators to push under one dispatcher for a source





- Sub queries for each source
  - DB_A

Select ?compound ?sideeffect
WHERE {
?compound sideEffect ?sideeffect. ?
compound contraindication
Hypovolemia
}

  - DB_B

Select ?compound ?formula WHERE
{
?compound formula ?formula
}

Section 4

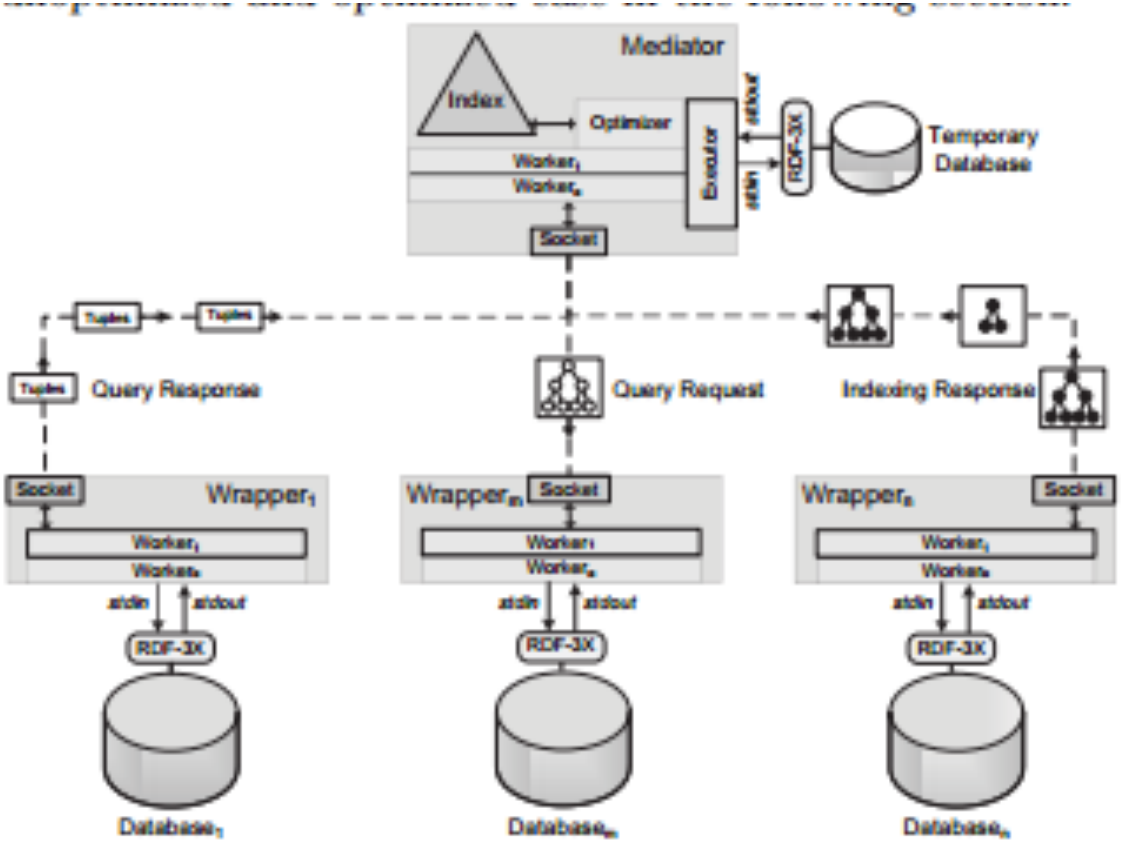
QUERY EXECUTION

Steps of the query execution model

1. Parse and optimize the query
2. Execute subqueries at the remote systems
3. Load local result into a global database
4. Execute the query on the global database

This is basically equivalent to executing the query on a relevant subgraph of the global graph. Therefore an RDF store can again be used as a global temporary database.

System Architecture

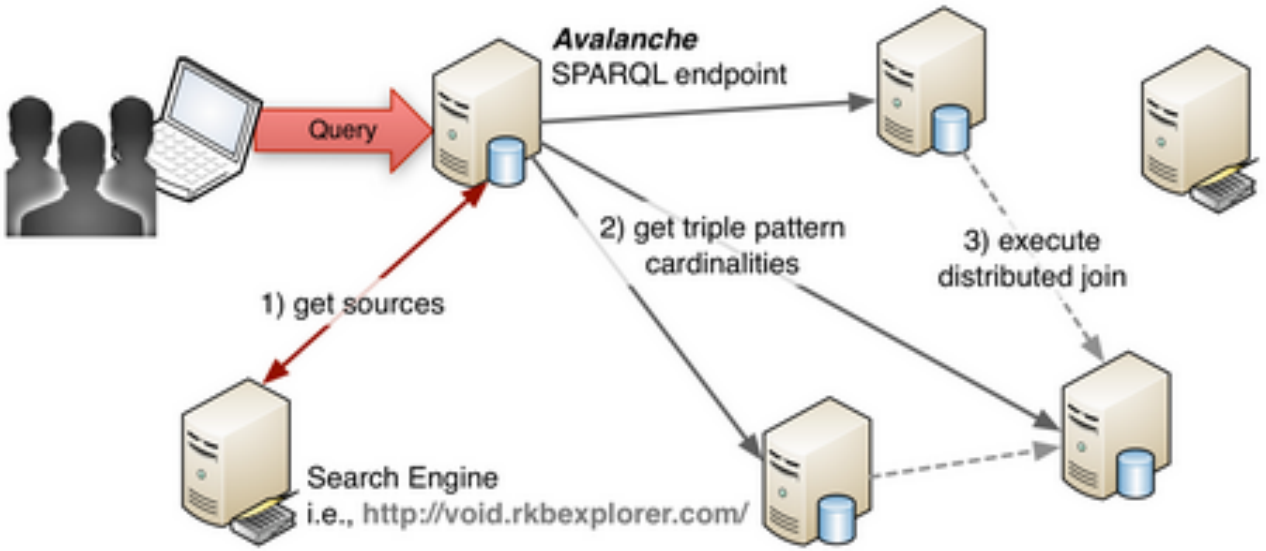


Section 5

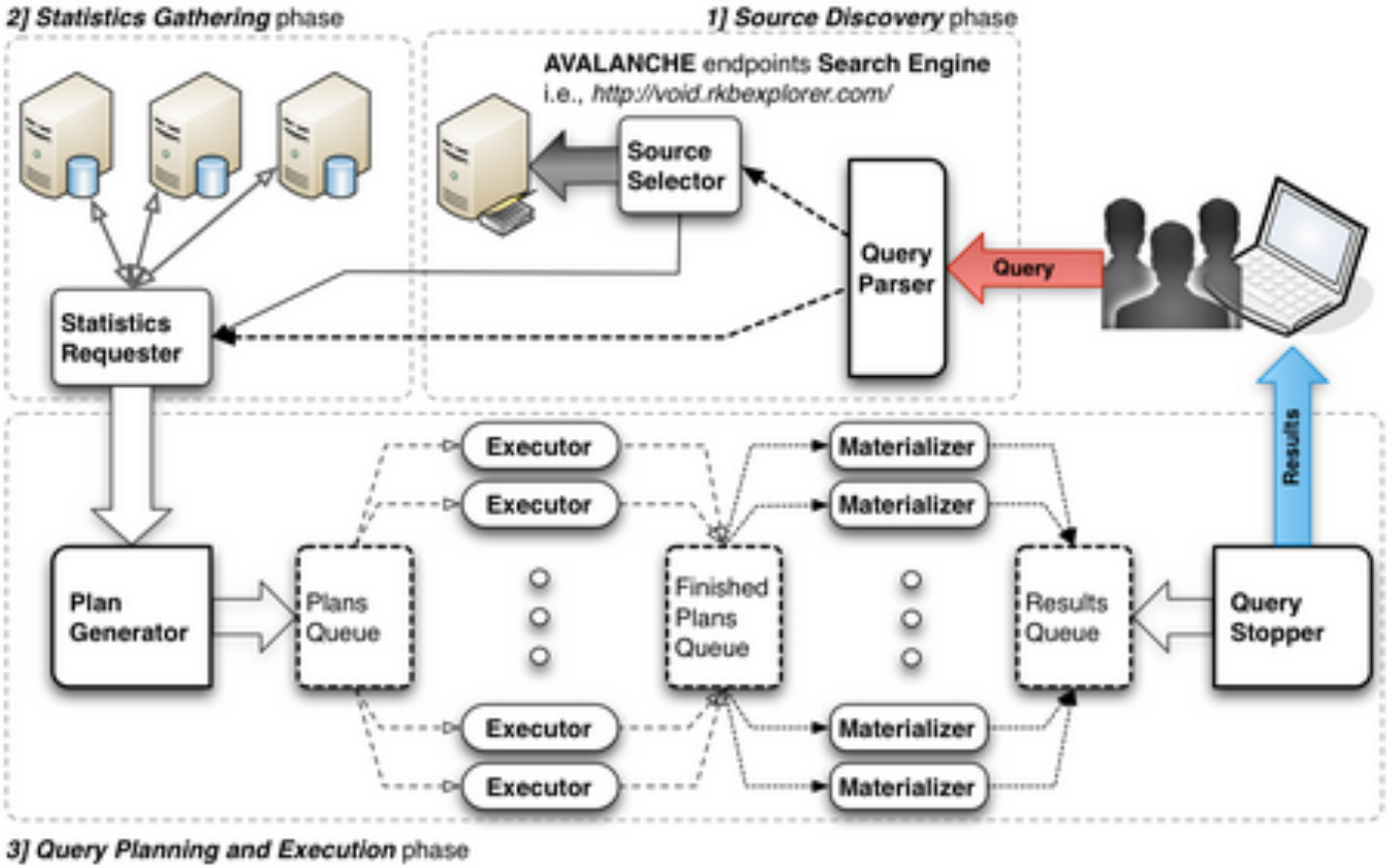
AVALANCHE

Avalanche

- Distributed SPARQL querying system without making any prior assumptions about the data distribution, schema-alignment, pertinent data statistics, data evolution, and data presence.



Query planning and execution in Avalanche



Discussion

- Indexing with vertical partitioning suggested in this work does nor perform well for datasets with large number of distinct predicates.
- Map reduce can be used to processing the results from subqueries.
- Read-only scenario only discussed in this work, update, delete and new indexes are not conidered.

References

This presentation based on the paper

Efficient Distributed Query Processing for RDF databases by Fabian Prasser, Alfons Kemper, Klaus A. Kuhn

THANK YOU