

Adaptive Semantic Data Management Techniques for Linked Data- Half Day Tutorial at ESWC 2011

Edna Ruckhaus and María-Esther Vidal

Universidad Simón Bolívar
Caracas, Venezuela

May 29th, 2011

Content

- 1 Introduction
- 2 Lecture 1 - The Optimize-then-Execute Paradigm
 - Cost-based optimization techniques
 - Physical operators
 - Traditional iterator model architecture
- 3 Lecture 2-Existing RDF Triple Stores
 - RDF Stores' Architectures
 - RDF triple stores-Remote Accessibility
 - Linked Data Query Processing
 - Semantic Data Management
 - Summary
- 4 Lecture 3-The Adaptive Query Processing Paradigm
- 5 Operators for Adaptive Join Processing
- 6 Routing based solutions
- 7 Inter-Operator Re-optimization
- 8 Summary

Introduction

Introduction-Traditional Data Management Systems[16, 23]

A Database Management System (DBMS) is a software designed to:

- consistently store and organize data,
- and to provide a secure and concurrent access to the data.

Introduction-Traditional Data Management Systems[16, 23]

A Database Management System (DBMS) is a software designed to:

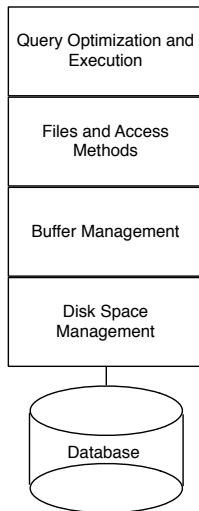
- consistently store and organize data,
 - Provide physical data structures to store large databases.
 - Stage large datasets between main memory and secondary storage.
 - Protect data from inconsistencies.
- and to provide a secure and concurrent access to the data.

Introduction-Traditional Data Management Systems[16, 23]

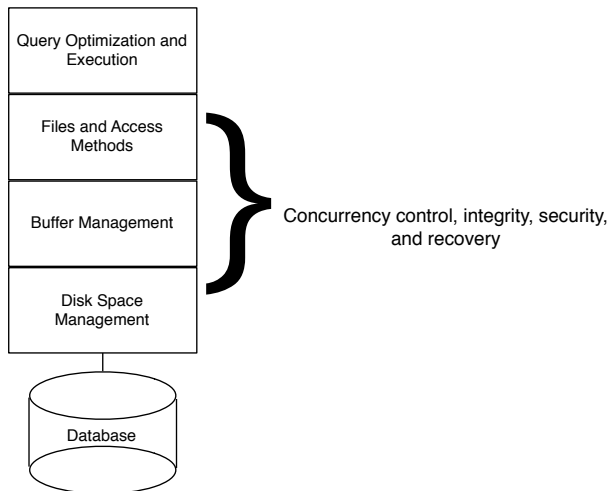
A Database Management System (DBMS) is a software designed to:

- consistently store and organize data,
 - Provide physical data structures to store large databases.
 - Stage large datasets between main memory and secondary storage.
 - Protect data from inconsistencies.
- and to provide a secure and concurrent access to the data.
 - Provide a query language.
 - Crash recovery.
 - Security and access control.

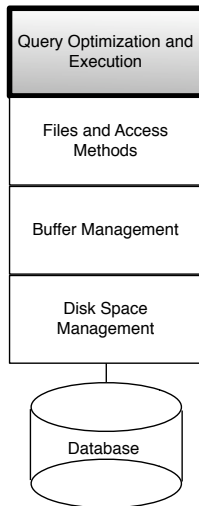
Traditional DBMS Architectures[16, 23]



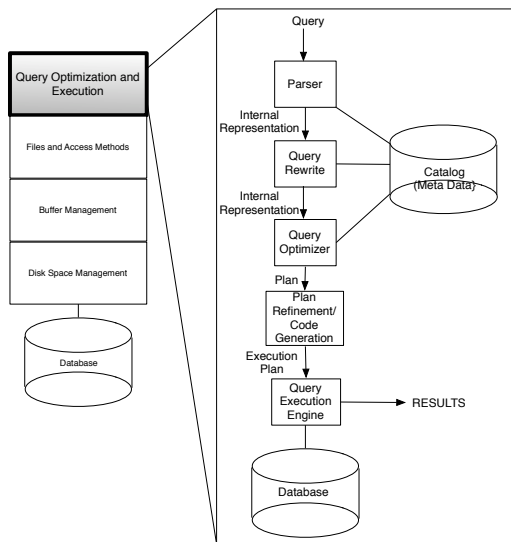
Traditional DBMS Architectures[16, 23]



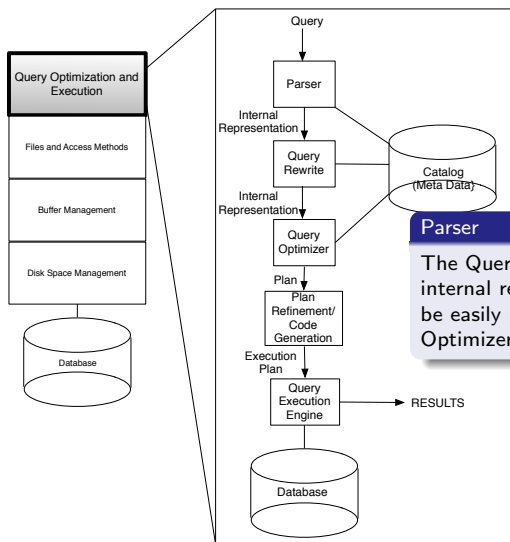
Traditional DBMS Architectures[16, 23]



Traditional DBMS Architectures[16, 23]



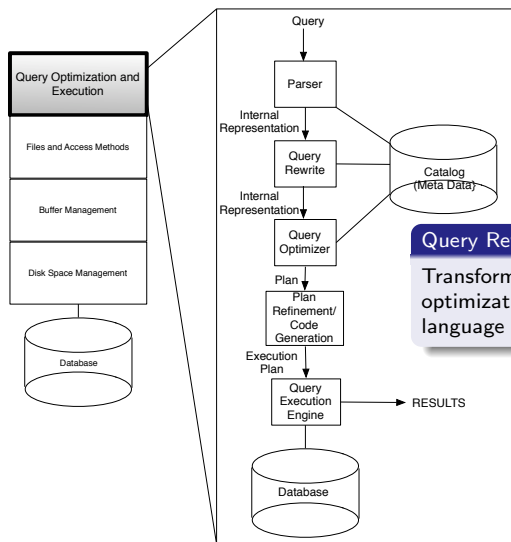
Traditional DBMS Architectures[16, 23]



Parser

The Query is parsed and translated into an internal representation, e.g., *Query Block* that can be easily processed by the Query Rewriter and Optimizer.

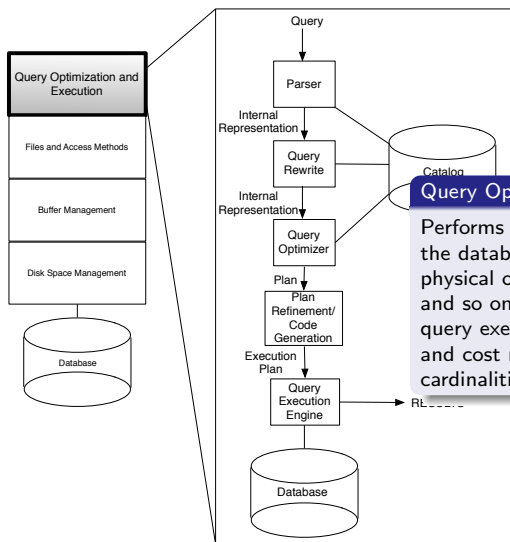
Traditional DBMS Architectures[16, 23]



Query Rewriter

Transforms a query to carry out heuristic optimization, based on algebraic properties of the language operators.

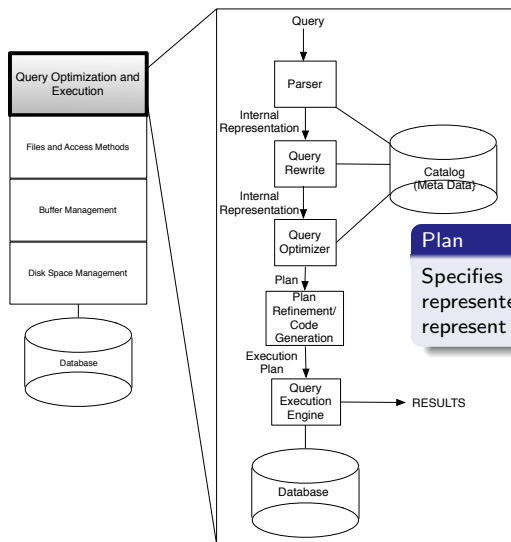
Traditional DBMS Architectures[16, 23]



Query Optimizer

Performs optimizations that depend on the size of the database, execution cost of the available physical operators, secondary storage structures, and so on; main memory to be assigned to the query execution is also decided. Data statistics and cost models are used to estimate costs and cardinalities.

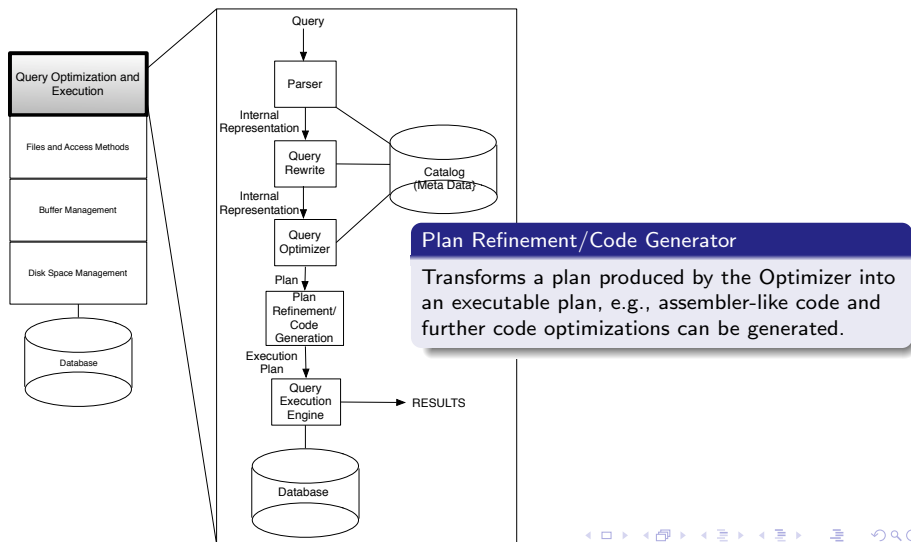
Traditional DBMS Architectures[16, 23]



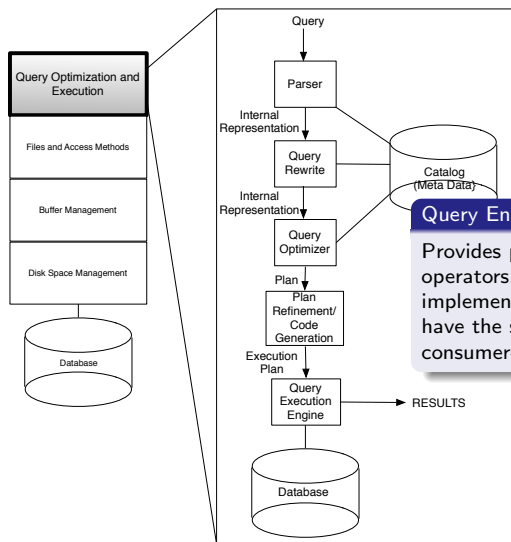
Plan

Specifies how the query will be executed; usually represented by trees, where internal nodes represent operators, and leaves data structures.

Traditional DBMS Architectures[16, 23]



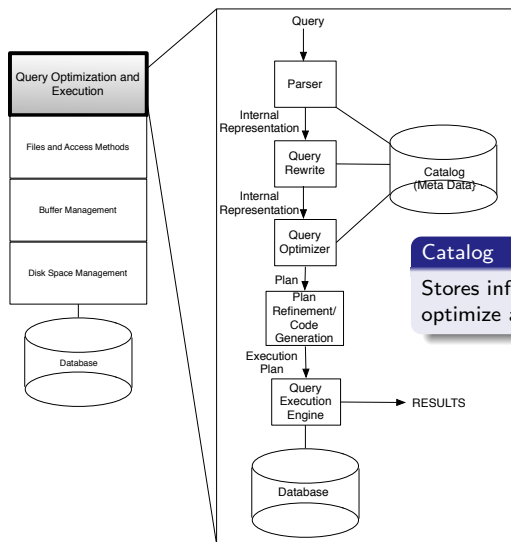
Traditional DBMS Architectures[16, 23]



Query Engine

Provides physical implementations for language operators. Usually an *iterator model* is implemented, where all the operators (iterators) have the same interface, and maintain a consumer-producer relationship.

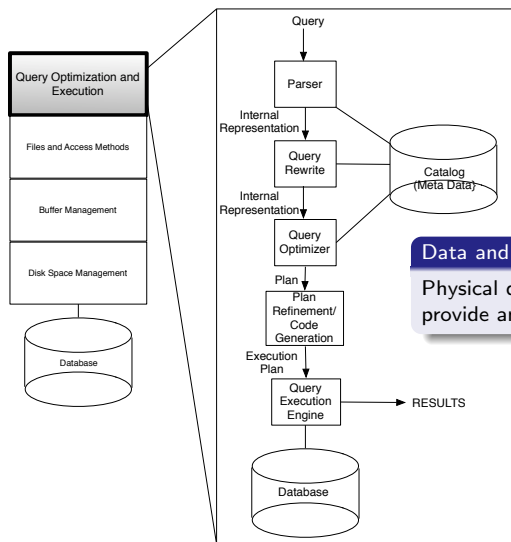
Traditional DBMS Architectures[16, 23]



Catalog

Stores information needed to parse, rewrite, and optimize a query.

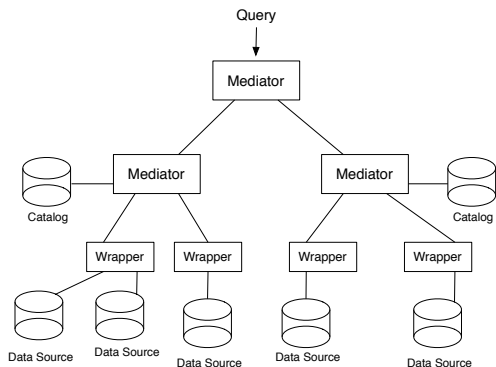
Traditional DBMS Architectures[16, 23]



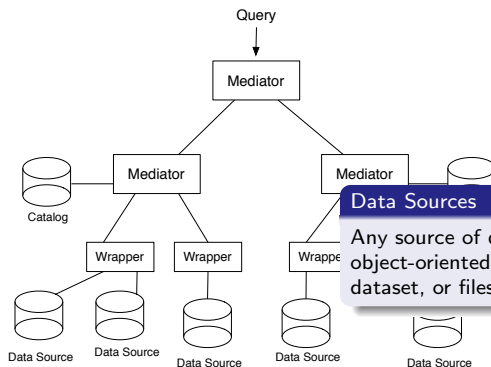
Data and Index Structures

Physical data secondary memory structures to provide an efficient access to the data.

Heterogeneous Database Systems- The Mediator and Wrapper Architecture[5, 24, 41, 43]



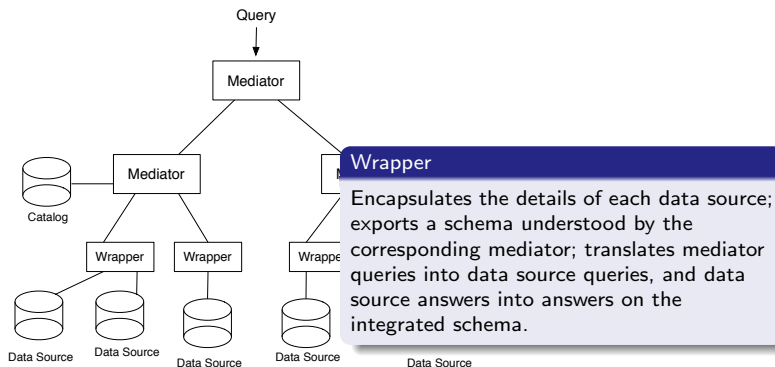
Heterogeneous Database Systems- The Mediator and Wrapper Architecture[5, 24, 41, 43]



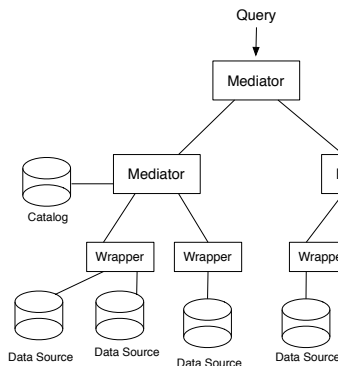
Data Sources

Any source of data, a relational, object-oriented or WWW database, an RDF dataset, or files.

Heterogeneous Database Systems- The Mediator and Wrapper Architecture[5, 24, 41, 43]



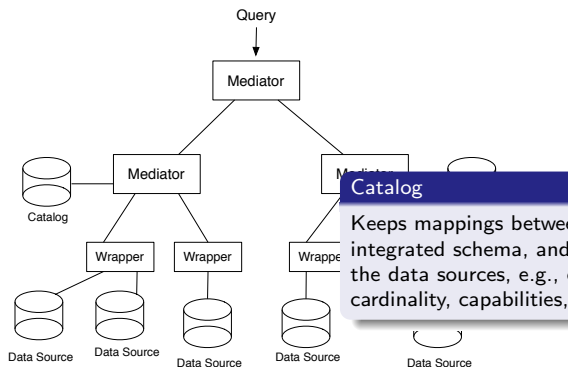
Heterogeneous Database Systems- The Mediator and Wrapper Architecture[5, 24, 41, 43]



Mediator

Exports an integrated schema, and hides properties of the heterogeneous sources; receives a query against the integrated schema, and rewrites, optimizes and executes the query on the sources; consults the catalog to retrieve definitions and statistics of the sources; gathers answers provided by the sources.

Heterogeneous Database Systems- The Mediator and Wrapper Architecture[5, 24, 41, 43]



Adaptive Query Processing [8, 13, 26]

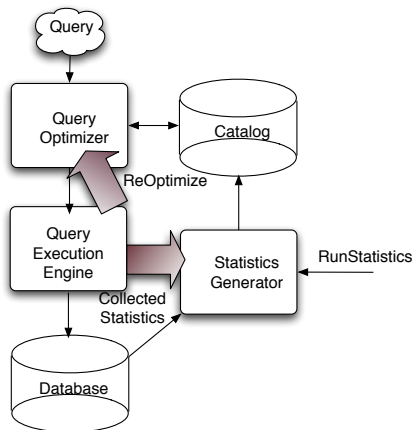
Adaptive Query Processing

Systems able to change their behavior by learning behavior of data providers.

- Receive information from the environment.
- Use up-to-date information to change their behavior.
- Keep iterating over time to adapt their behavior based on the environment conditions.

Adaptivity is needed:

- Misestimated or missing statistics.
- Unexpected correlations.
- Unpredictable costs.
- Dynamically changing data, workload and sources availability.
- Rates at which tuples arrive from the sources, change.



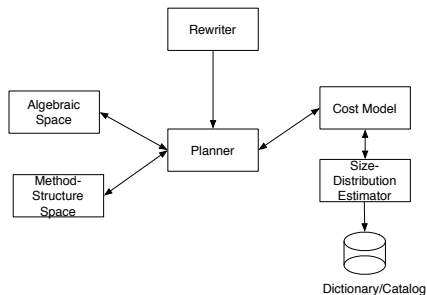
Lecture 1: The Optimize-then-Execute Paradigm

The Optimize-then-Execute Paradigm

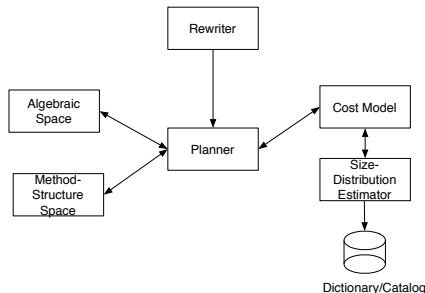
Traditional Query Processing techniques:

- Parse a declarative query.
- Generate an intermediate representation of the query.
- Produce an efficient logical and physical plan.
- Execute the query plan without making runtime decisions.

Query Optimizer Architecture[16, 23]



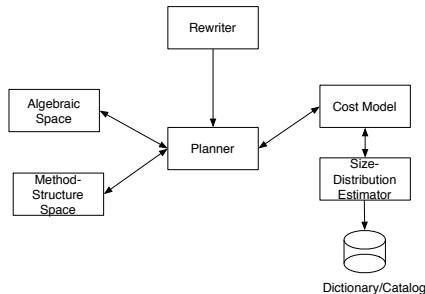
Query Optimizer Architecture[16, 23]



Rewriter

Applies algebraic transformations to the query to produce a more efficient equivalent query; some transformations are: flatten out queries, using views, etc. Nested queries are decomposed into *query blocks*.

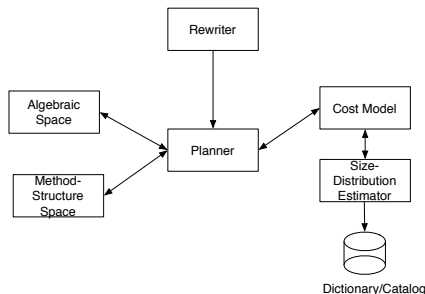
Query Optimizer Architecture[16, 23]



Planner

- Traverses the space of possible execution plans following a particular search strategy, e.g., dynamic programming, randomized algorithms.
- *Query blocks* are optimized one at a time.
- All available access methods are considered for each relation.
- All the ways to join the relations one-at-a-time; permutations of relations and different join methods are considered.

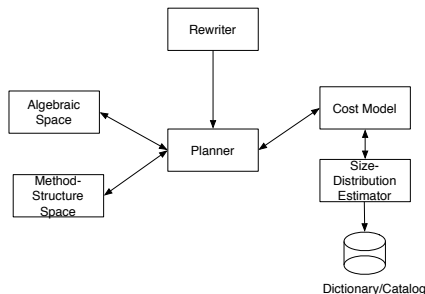
Query Optimizer Architecture[16, 23]



Algebraic Space

Set of algebraic rules that restrict the space of plans transformations, and guide the planner into the space of efficient logical plans.

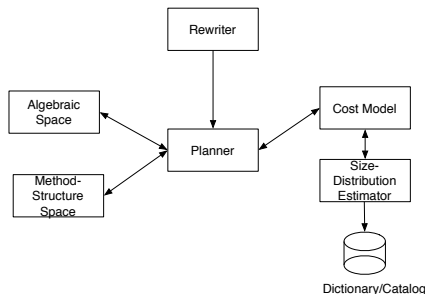
Query Optimizer Architecture[16, 23]



Method-Structure Space

Set of rules that restrict the space of physical implementations of each logical plan, and guide the planner into the space of efficient physical plans.

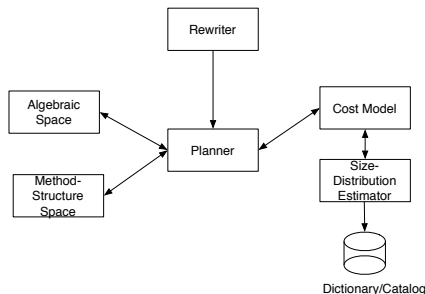
Query Optimizer Architecture[16, 23]



Cost Model

Set of arithmetic rules or statistical techniques, to estimate the execution cost and cardinality of logical and physical plans.

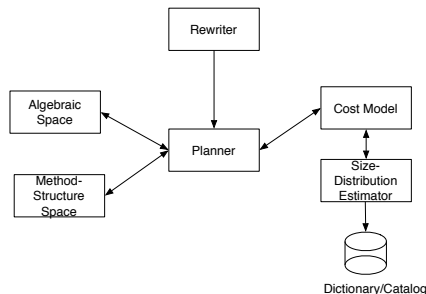
Query Optimizer Architecture[16, 23]



Size-Distribution Estimator

Set of arithmetic rules or statistical techniques to estimate the cardinalities of dataset to be accessed as well as its distributions, and the selectivity of a query select condition.

Query Optimizer Architecture[16, 23]



Dictionary

Set of statistics that characterize the dataset, e.g., number of different values of an attribute, or different subjects, properties, or values; distributions, etc. Stored statistics depend on the type of size-distribution estimator.

Algebraic Space

Set of rules that prune the space of possible query plans:

- Selections and projections are pushed ahead of "joins".
- Cross products are never formed, unless the query itself asks for them.
- Inner operand of each join is a database relation, never an intermediate result (*left-linear plans*).
 - Allow output of each operator to be pipelined into the next operator without storing it in a temporary relation.

Algebraic Space

Set of rules that prune the space of possible query plans:

- Selections and projections are pushed ahead of "joins".
- Cross products are never formed, unless the query itself asks for them.
- Inner operand of each join is a database relation, never an intermediate result (*left-linear plans*).
 - Allow output of each operator to be pipelined into the next operator without storing it in a temporary relation.
 - Preexisting indices can be used for inner database relations.

Selection Ordering: Search Space

- Problem of determining the order in which to apply commutative selection predicates (or filters) $S_1, \dots, S_n$.
- The order of applying the selections, will depend on the cost and the probability that a triple or tuple satisfies the predicates, $cost(S_1, \dots, S_n)$ and $p(S_1 \wedge \dots \wedge S_n)$, respectively.
- If a tuple is known to satisfy a set of predicates $\{S_1, \dots, S_i\}$, the conditional probability of also satisfying S_{i+1} is denoted by $p(S_{i+1} | S_1, \dots, S_i)$.

Selection Ordering: Search Space

- Problem of determining the order in which to apply commutative selection predicates (or filters) $S_1, \dots, S_n$.
- The order of applying the selections, will depend on the cost and the probability that a triple or tuple satisfies the predicates, $cost(S_1, \dots, S_n)$ and $p(S_1 \wedge \dots \wedge S_n)$, respectively.
- If a tuple is known to satisfy a set of predicates $\{S_1, \dots, S_i\}$, the conditional probability of also satisfying S_{i+1} is denoted by $p(S_{i+1} | S_1, \dots, S_i)$.
 - **Independent assumption:** $p(S_{i+1} | S_1, \dots, S_i) = p(S_{i+1})$, and the optimal order can be found in $O(n \log(n))$ time.
 - **Correlated Predicates:** finding an optimal conditional plan for a given dataset D is NP-Hard. Some solutions have been implemented as Bayesian networks[15, 31].

Selection Ordering: Search Space

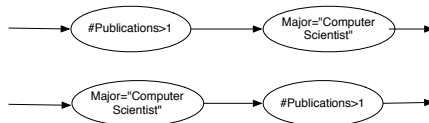
Example

Select ID From STUDENT
Where #Publications>1 and
Major="Computer Scientist"

Selection Ordering: Search Space

Example

Select ID From STUDENT
Where #Publications>1 and
Major="Computer Scientist"

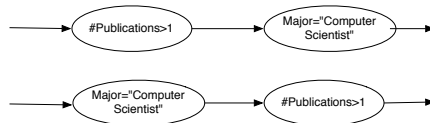


Two Serial Plans

Selection Ordering: Search Space

Example

Select ID From STUDENT
Where $\#Publications > 1$ and
Major="Computer Scientist"



Two Serial Plans



A Conditional Plan

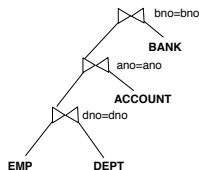
Join Ordering: Search Space

- Defined based on algebraic rules established by the *Algebraic Space* module and physical operator methods offered by the *Method-Structure Space* module.

Join Ordering: Search Space

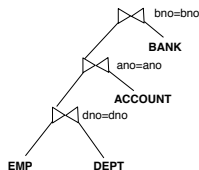
- Defined based on algebraic rules established by the *Algebraic Space* module and physical operator methods offered by the *Method-Structure Space* module.
- Plan space can be too large, so strategies to traverse the space are required.

Join Ordering: Search Space

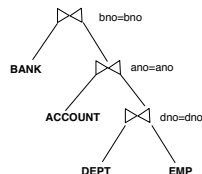


Left-linear plan

Join Ordering: Search Space

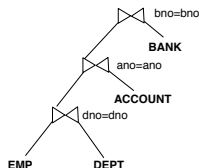


Left-linear plan

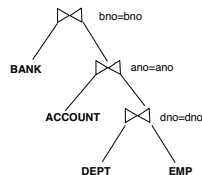


Right-linear plan

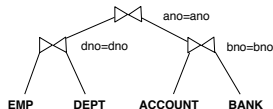
Join Ordering: Search Space



Left-linear plan



Right-linear plan



Bushy Plan

Search Strategies

- Set of meta-heuristics used to explore the set of alternative execution plans.
- Guided by the *Algebraic* and *Method-Structure* space modules.
- Find the cheapest plan based on *Cost Model* and the *Size-Distribution Estimator*.
- Some strategies: *dynamic programming*, *randomized* algorithms.

Dynamic Programming

- Performs a dynamically pruning and exhaustive search.
- Constructs all alternative join trees, guided by the *Algebraic* space module.
- Iterates on the relations joined so far.
- Prunes sub-optimal solutions.
- May consider *interesting orders*.

Dynamic Programming

Algorithm 2.1: DYNPROGRAM($Rels$)

comment: $Rels$: Set of relations

comment: Output: linear left plan for $Rels$.

```
PartialSolutions  $\leftarrow$  All possible ways to access simple relations  
Remove from PartialSolutions subOptimal solutions.  
for  $i \leftarrow 2$  to  $|Rels|$   
  do {  
    for each  $pt \in \text{PartialSolutions}$   
      do for each  $R_i \in Rels$   
      do  $pt \leftarrow (pt \bowtie R_i)$   
      Remove from PartialSolutions subOptimal solutions.  
  }  
return (one plan from PartialSolutions)
```

Randomized Algorithms

- Random walks are performed to traverse the space of plans.
- Transformations are conducted to move from a plan to the other.
- Each visited plan is associated with a cost.
- The objective is to reach a plan with a global minimum cost.

Randomized Algorithms

Algorithm 2.2: SIMULATEDANNEALING(*State*, *Rels*)

comment: *State*: Random starting point; *Rels*: Set of relations

comment: Output: Bushy plan for *Rels*.

```

MinState ← State
MinCost ← cost(State)
Cost ← MinCost
t ← "Starting temperature"
repeat
  repeat
    NewState ← "State after random move"
    NewCost ← cost(NewState)
    if (NewCost < Cost)
      then { State ← NewState
            Cost ← NewCost
          }
      else "with probability  $e^{NewCost - Cost / t}$ "
      { State ← NewState
        Cost ← NewCost
      }
    if (Cost < MinCost)
      then { MinState ← State
            MinCost ← Cost
          }
  until "equilibrium is reached"
  "Reduce temperature"
until "frozen"
return (MinState)

```

Access Paths

- Methods for retrieving the tuples that satisfy a given condition.
- Tree index matches terms with attributes in a prefix of the search key.
- A hash index matches terms ($\text{attribute} = \text{value}$) for every attribute in the search key of the index.

Access Paths

- Methods for retrieving the tuples that satisfy a given condition.
- Tree index matches terms with attributes in a prefix of the search key.
 - tree index on $\langle a, b, c \rangle$ matches conjunctions of terms with attributes in prefixes of a, b, c .
- A hash index matches terms (attribute = value) for every attribute in the search key of the index.

Access Paths

- Methods for retrieving the tuples that satisfy a given condition.
- Tree index matches terms with attributes in a prefix of the search key.
- A hash index matches terms (attribute = value) for every attribute in the search key of the index.
 - Hash index on $\langle a, b, c \rangle$ matches conjunctions of equality conditions, on a , b , and c .

Projections

- Removes duplicates, the most expensive operation.
- Sort based operator: sort in terms of the attributes to be projected, and remove duplicates.
- Hash based operator: hashes in terms of the attributes to be projected, to partition the tuples; loads partitions into a hash table, and remove duplicates.
- If there exist indices on the attributes to be projected, sorting may be cheaper.

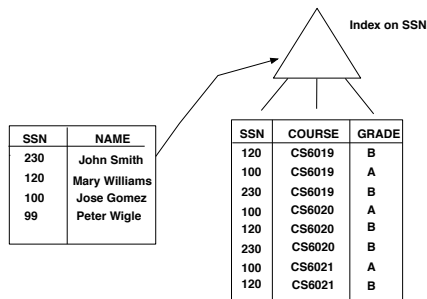
Join Index Nested Loop

- Indices exist on the join attributes of the inner relation of the join.
- Cost depends on the type of index and on the clustering:

Join Index Nested Loop

- Indices exist on the join attributes of the inner relation of the join.
- Cost depends on the type of index and on the clustering:
 - To find matching tuples: clustered index: 1 I/O, unclustered: up to 1 I/O per matching tuple.

Example-Join Index Nested Loop



Index Nested Loop Join by SSN

Sort Merge Join

- Sorts input tables on the join attributes;
- Scans sorted tables;
- Merges on join attributes.
- Cost is optimal in case input tables are sorted.

Example Sort Merge Join

SSN	NAME
230	John Smith
120	Mary Williams
100	Jose Gomez
99	Peter Wigle

SSN	COURSE	GRADE
120	CS6019	B
100	CS6019	A
230	CS6019	B
100	CS6020	A
120	CS6020	B
230	CS6020	B
100	CS6021	A
120	CS6021	B

Sort Merge by SSN

Example Sort Merge Join

SSN	NAME
230	John Smith
120	Mary Williams
100	Jose Gomez
99	Peter Wigle

SSN	COURSE	GRADE
120	CS6019	B
100	CS6019	A
230	CS6019	B
100	CS6020	A
120	CS6020	B
230	CS6020	B
100	CS6021	A
120	CS6021	B

SSN	NAME
99	Peter Wigle
100	Jose Gomez
120	Mary Williams
230	John Smith

SSN	COURSE	GRADE
100	CS6019	A
100	CS6020	A
100	CS6021	A
120	CS6019	B
120	CS6020	B
120	CS6021	B
230	CS6020	B
230	CS6019	B

Sort Merge by SSN

Sorting by SSN

Example Sort Merge Join

SSN	NAME
230	John Smith
120	Mary Williams
100	Jose Gomez
99	Peter Wigle

SSN	COURSE	GRADE
120	CS6019	B
100	CS6019	A
230	CS6019	B
100	CS6020	A
120	CS6020	B
230	CS6020	B
100	CS6021	A
120	CS6021	B

SSN	NAME
99	Peter Wigle
100	Jose Gomez
120	Mary Williams
230	John Smith

SSN	COURSE	GRADE
100	CS6019	A
100	CS6020	A
100	CS6021	A
120	CS6019	B
120	CS6020	B
120	CS6021	B
230	CS6020	B
230	CS6019	B

Sort Merge by SSN

Sorting by SSN

SSN	COURSE	GRADE	NAME
100	CS6019	A	Jose Gomez
100	CS6020	A	Jose Gomez
100	CS6021	A	Jose Gomez
120	CS6019	B	Mary Williams
120	CS6020	B	Mary Williams
120	CS6021	B	Mary Williams
230	CS6020	B	John Smith
230	CS6019	B	John Smith

Merging by SSN

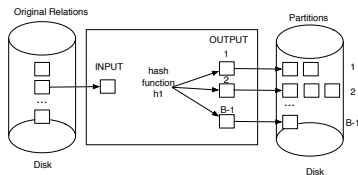
Hash Join

- Two fold physical operator:

Partition: input tables are partitioned using a hash function h_1 ; only tuples in the same partitions of two tables are joined.

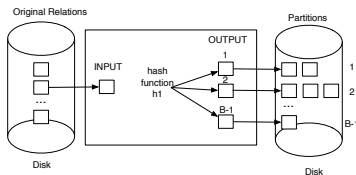
Probing: each partition of the outer table is loaded in a hash table using a hash function h_2 ; the corresponding partition of the inner table is scanned to search for matches.

Example Hash Join

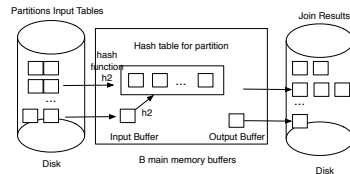


Partition Phase

Example Hash Join



Partition Phase



Probing Phase

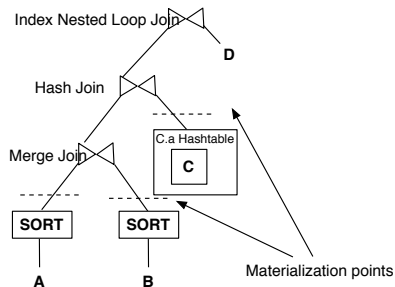
Execution Engine-Blocking Operators

- Require more than one pass over the input to create the output.
- Hold intermediate state to compute the output.
- Some blocking operators:

Execution Engine-Blocking Operators

- Require more than one pass over the input to create the output.
- Hold intermediate state to compute the output.
- Some blocking operators:
 - Project:** must sort and then merge the input.
 - Hash Join:** must partition the input, temporary store the input in a hash table, probe hashed data many times.

Example-Blocking Operators



Blocking Operators

Execution Engine-Pipelined Plans

- Executes all operators in the plan in parallel.
- Operators or *iterators*, consume tuples from its children and output produced tuples to their parents.
- Tuples flow from the leaf nodes towards the root of the tree.
- *Iterators* have the following functions:
 - Open** prepares an operator for producing data.
 - Next** produces an item.
 - Close** performs final house-keeping.

Examples of Iterator Functions [16]

Iterator	<i>Open</i>	<i>Next</i>	<i>Close</i>
Scan	<i>open</i> input	call <i>next</i> on input	<i>close</i> input

Examples of Iterator Functions [16]

Iterator	<i>Open</i>	<i>Next</i>	<i>Close</i>
Scan	<i>open</i> input	call <i>next</i> on input	<i>close</i> input
Select	<i>open</i> input	call <i>next</i> on input until an item qualifies	<i>close</i> input

Examples of Iterator Functions [16]

Iterator	<i>Open</i>	<i>Next</i>	<i>Close</i>
Scan	<i>open</i> input	call <i>next</i> on input	<i>close</i> input
Select	<i>open</i> input	call <i>next</i> on input until an item qualifies	<i>close</i> input
Hash Join	allocate hash directory open left <i>partition</i> input build hash table <i>next</i> on <i>partition</i> input <i>close partition</i> input <i>open</i> right <i>probe</i> input	call <i>next</i> on <i>probe</i> input until a match is found	<i>close probe</i> input deallocate hash directory

Examples of Iterator Functions [16]

Iterator	<i>Open</i>	<i>Next</i>	<i>Close</i>
Scan	<i>open</i> input	call <i>next</i> on input	<i>close</i> input
Select	<i>open</i> input	call <i>next</i> on input until an item qualifies	<i>close</i> input
Hash Join	allocate hash directory open left <i>partition</i> input build hash table <i>next</i> on <i>partition</i> input <i>close partition</i> input <i>open right probe</i> input	call <i>next</i> on <i>probe</i> input until a match is found	<i>close probe</i> input deallocate hash directory
Merge Join	<i>open</i> both inputs	get <i>next</i> item from input with smaller key until a match is found	<i>close</i> both inputs

Lecture 2-Existing RDF Triple Stores

Features of RDF Storage Systems

Architecture database engines to store, access and manage RDF data.

Secondary Memory Structures to provide an efficient storage and access to RDF data.

Accessibility ability to access data locally or remotely stored.

Link Dereferencing capability of dereferencing RDF data URLs.

Adaptivity ability to adapt query execution schedulers to data availability.

Semantic Data Management query processing techniques that use data based on their meaning.

Typical RDF Stores' Architectures:

DBMS-backed stores make use of existing DBMS to store and access RDF data.

Native stores implement an entire database engine to store and access RDF data.

Hybrid Approaches combine both native and DBMS-backed architectural styles to store and access RDF data.

DBMS-backed stores

- The RDF stores are built on top of an existing DBMS.
- Query execution techniques as well as physical operators and optimization strategies may be implemented to exploit them implemented RDF database.
- Several approaches:

DBMS-backed stores

- The RDF stores are built on top of an existing DBMS.
- Query execution techniques as well as physical operators and optimization strategies may be implemented to exploit them implemented RDF database.
- Several approaches:
 - A **giant triples** table, and up to six indices on combinations of *subject*, *property* and *object*; e.g., Hexastore[40], YARS2[18].
 - Tables for **clustered predicates**, where predicates on the same entity class or correlated predicates are stored in the same table; left over additional infrequent predicates are stored in the same table; e.g., Jena¹, Sesame².
 - A **vertically partitioning** design, where there is a two column table for each property and column-oriented DBMS are used to implement the tables , e.g., MonetDB[22], C-store[33].

¹<http://jena.sourceforge.net/ontology/>

²<http://www.openrdf.org/>

Triple-Store Schema

Subject	Property	Object
_:id0	type	Term
_:id0	forOffice	AZ
AZ	type	Office
_:id1	sponsor	HR45
Wigle	sponsor	HR44
Smith	sponsor	HR44
Wigle	supported	SJ44
Rush	sponsor	HR45
_:id1	supported	SJ37
_:id5	sponsor	HR50
_:id1	forOffice	MD
_:id3	type	Term
_:id3	forOffice	MD
MD	type	Office
_:id4	supported	SJ38
_:id4	sponsor	HR51
_:id1	votes	Bill125
_:id1	votes	Bill126
_:id1	votes	Bill127
_:id1	votes	Bill128

- Triples are stored in a single, and giant table with three attributes <subject, predicate, object>.
- One row for each triple.
- Processing a query may involve the execution of a large number of self-joins.

Clustered Property Tables

- Denormalization of RDF tables by physically storing them in a wider and flattened representation.
- Properties that tend to be defined together are grouped into clusters and stored in a property table.
- Multiple property tables with different clusters of properties can be created.
- The same property can appear at most in one property table.
- Class-specific schema with attributes named after the corresponding RDF predicates.
- Attributes can range from one predicate to all the predicates of the same entity type.

Clustered Property Tables

- NULL values represent that a property is not defined for a certain subject.
- Multi-valued attributes are difficult to represent; list, set or bag attributes need to be used.
- If properties are bound in a query, property tables require fewer joins because self-joins on the subject column can be eliminated, however, there is a trade off between query complexity and table sparsity.
- Query processing complexity can be very high if the property is unbound in the query, i.e., all flattened tables will have to be queried and combined by using union or join.

Clustered Property Tables

Subject	Property	Object
_:id0	type	Term
_:id0	forOffice	AZ
AZ	type	Office
_:id1	sponsor	HR45
Wigle	sponsor	HR44
Smith	sponsor	HR44
Wigle	supported	SJ44
Rush	sponsor	HR45
_:id1	supported	SJ37
_:id5	sponsor	HR50
_:id1	forOffice	MD
_:id3	type	Term
_:id3	forOffice	MD
MD	type	Office
_:id4	supported	SJ38
_:id4	sponsor	HR51
_:id1	votes	Bill125
_:id1	votes	Bill126
_:id1	votes	Bill127
_:id1	votes	Bill128

Clustered Property Tables

Subject	Property	Object
_.id0	type	Term
_.id0	forOffice	AZ
AZ	type	Office
_.id1	sponsor	HR45
Wigle	sponsor	HR44
Smith	sponsor	HR44
Wigle	supported	SJ44
Rush	sponsor	HR45
_.id1	supported	SJ37
_.id5	sponsor	HR50
_.id1	forOffice	MD
_.id3	type	Term
_.id3	forOffice	MD
MD	type	Office
_.id4	supported	SJ38
_.id4	sponsor	HR51
_.id1	votes	Bill125
_.id1	votes	Bill126
_.id1	votes	Bill127
_.id1	votes	Bill128

Property Table

Subject	forOffice	sponsor	supported
_.id0	AZ	NULL	NULL
_.id1	MD	HR45	SJ37
_.id3	MD	NULL	NULL
_.id4	NULL	HR51	SJ38
Wigle	NULL	HR44	SJ44
Smith	NULL	HR44	NULL
Rush	NULL	HR45	NULL

Left-Over Triples

Subject	Property	Object
AZ	type	Office
MD	type	Office
_.id0	type	Term
_.id3	type	Term

Multi-valued Property Table
votes

Subject	Object
_.id1	Bill125
_.id1	Bill126
_.id1	Bill127
_.id1	Bill128

Clustered Property Tables, sorted by subjects.

Vertically Partitioned Approach [1, 2, 22, 33]

- A Triples table is decomposed into n two column tables, where n is the number of properties.

Vertically Partitioned Approach [1, 2, 22, 33]

- A Triples table is decomposed into n two column tables, where n is the number of properties.
 - first column contains the subjects.
 - second column contains the object values.
- Tables are sorted by subjects.
- Indices are created on the two columns.

Vertical Partition Table- Advantages

Tables are sorted by subject: efficient merge join implementations are supported.

Multi-valued attributes: a subject can have more than one value for a particular property.

Semi-structured records: subjects may define different properties.

I/O reduction: only properties used in a query are accessed.

Simple partitioning criteria: clustering algorithms are not required.

Faster Joins: than self joins between a giant table or unions and joins between clustered tables .

Vertical Partitions Tables

Subject	Property	Object
_:id0	type	Term
_:id0	forOffice	AZ
AZ	type	Office
_:id1	sponsor	HR45
Wigle	sponsor	HR44
Smith	sponsor	HR44
Wigle	supported	SJ44
Rush	sponsor	HR45
_:id1	supported	SJ37
_:id5	sponsor	HR50
_:id1	forOffice	MD
_:id3	type	Term
_:id3	forOffice	MD
MD	type	Office
_:id4	supported	SJ38
_:id4	sponsor	HR51

Giant Table; one row for each triple.

Vertical Partitions Tables

Subject	Property	Object
_:id0	type	Term
_:id0	forOffice	AZ
AZ	type	Office
_:id1	sponsor	HR45
Wigle	sponsor	HR44
Smith	sponsor	HR44
Wigle	supported	SJ44
Rush	sponsor	HR45
_:id1	supported	SJ37
_:id5	sponsor	HR50
_:id1	forOffice	MD
_:id3	type	Term
_:id3	forOffice	MD
MD	type	Office
_:id4	supported	SJ38
_:id4	sponsor	HR51

type

Subject	Object
_:id0	Term
AZ	Office
_:id3	Term
MD	Office

forOffice

Subject	Object
_:id0	AZ
_:id1	MD
_:id3	MD

sponsor

Subject	Object
_:id1	HR45
Wigle	HR44
Smith	HR44
Wigle	HR44
Rush	HR45
_:id1	HR50
_:id4	HR51

supported

Subject	Object
_:id1	SJ37
_:id4	SJ38

Property Tables, sorted by subjects.

Giant Table; one row for each triple.

Vertical Partition Table-Column Oriented

- Tables are stored as collections of columns
- Header information is stored in separate columns and can be ignored.
- Strings are encoded in a mapping dictionary, and attributes in the two column tables are fixed-length that are stored as arrays.
- Compression algorithms are used to compress each column.
- Direct access to sorted files.

Native store

- Secondary memory structures are implemented to store RDF data.
- Query engines and physical operators are designed to exploit the properties of the RDF data structures.
- Query optimization techniques are tailored for the RDF store query engine, physical operators and data structures.

³<http://www.openjena.org/TDB/>

⁴<http://www.franz.com/agraph/allegrograph/>

Native store

- Secondary memory structures are implemented to store RDF data.
- Query engines and physical operators are designed to exploit the properties of the RDF data structures.
- Query optimization techniques are tailored for the RDF store query engine, physical operators and data structures.
- Some native RDF stores: RDF-3X[29], Jena TDB³, AllegroGraph⁴, BitMat[6].

³<http://www.openjena.org/TDB/>

⁴<http://www.franz.com/agraph/allegrograph/>

RDF-3X. [29]

- RISC style operators.
- Triples Table, each row represents a triple.
- Mapping Dictionary, replacing all the literal strings by an id; two dictionary indices.
- Compressed clustered B+-tree to index all triples:

RDF-3X. [29]

- RISC style operators.
- Triples Table, each row represents a triple.
- Mapping Dictionary, replacing all the literal strings by an id; two dictionary indices.
- Compressed clustered B+-tree to index all triples:
 - Six permutations of subject, predicate and object; each permutation in a different index.
 - Triples are sorted lexicographically in each B+-tree.

RDF-3X

Subject	Property	Object
_:id0	type	Term
_:id0	forOffice	AZ
AZ	type	Office
_:id1	sponsor	HR45
Wigle	sponsor	HR44

Triples Table.

RDF-3X

Subject	Property	Object
_:id0	type	Term
_:id0	forOffice	AZ
AZ	type	Office
_:id1	sponsor	HR45
Wigle	sponsor	HR44

Triples Table.

Triples Table

Subject	Property	Object
0	4	7
0	5	1
1	4	8
2	6	9
3	6	10

Dictionary Table

ID	Value
0	_:id0
1	AZ
2	_:id1
3	Wigle
4	type
5	forOffice
6	sponsor
7	Term
8	Office
9	HR45
10	HR44

Encoded Triples Table and the Dictionary Table.

RDF-3X. [29]

- Compression schema for triples where only changes between triples are stored.
- Compression is done at byte-level.
- Compute the delta for each value and then, use the minimal number of bytes to encode the delta.

RDF-3X. [29]

- Compression schema for triples where only changes between triples are stored.
- Compression is done at byte-level.
- Compute the delta for each value and then, use the minimal number of bytes to encode the delta.

Header Byte denotes the number of bytes used by values v1, v2 and v3.

Delta consumes between 0 and 4 bytes,

Gap Bit small gap, when only v3 changes, and the delta is less than 128, this number is stored in the header byte.

RDF-3X. [29]

- Compression schema for triples where only changes between triples are stored.
- Compression is done at byte-level.
- Compute the delta for each value and then, use the minimal number of bytes to encode the delta.

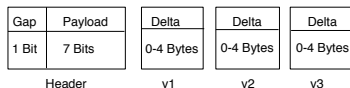
Header Byte denotes the number of bytes used by values v_1 , v_2 and v_3 .

Delta consumes between 0 and 4 bytes,

Gap Bit small gap, when only v_3 changes, and the delta is less than 128, this number is stored in the header byte.

- Efficient scan if prefix is known; each triple pattern can be answered by a single range scan.

RDF-3X-Compressed Indices



Compressed Index Structure.

RDF-3X-Compressed Indices

Gap	Payload	Delta	Delta	Delta
1 Bit	7 Bits	0-4 Bytes	0-4 Bytes	0-4 Bytes
Header		v1	v2	v3

Compressed Index Structure.

Algorithm 3.1: COMPRESS(((v1, v2, v3), (prev1, prev2, prev3)))

comment: output: Header Byte, Deltav1, Deltav2, Deltav3

```

if (v1 = prev1 ∧ v2 = prev2)
{
  if (v3 - prev3 < 128)
    return ((v3 - prev3, 0, 0, 0))
  else return (encode(0, 0, v3 - prev3 - 128))
  else if (v1 = prev1)
    return (encode(0, v2 - prev2, v3))
  else
    return (encode(v1 - prev1, v2, v3))

```

Computes the delta for each value.

v1=123 v2=24 v3=34 prev1=123 prev2=20 prev3=30
Header=154 Deltav1=0 Deltav2=4 Deltav3=34

Gap	Payload	Delta	Delta	Delta
1	0011010	0	11	100010
Header		v1	v2	v3

Compressed Index Example.

Algorithm 3.2: ENCODE((v1, v2, v3))

```

Header ← 128 + bytes(v1) * 52 + bytes(v2) * 51 + bytes(v3) * 50
Deltav1 ← nonzero tail bytes of v1
Deltav2 ← nonzero tail bytes of v2
Deltav3 ← nonzero tail bytes of v3
return (Header, Deltav1, Deltav2, Deltav3)

```

Writes the header and deltas in the compressed triple.

RDF-3X-Aggregated Indices

- Each pair of a triple entry, indexes an aggregated count or number of occurrences of this pair in the triples set.
- Six indices on all the combinations of two entries are created.
- Aggregated number is required to produce the correct number of duplicates of a query.

RDF-3X-Aggregated Indices

- Each pair of a triple entry, indexes an aggregated count or number of occurrences of this pair in the triples set.
- Six indices on all the combinations of two entries are created.
- Aggregated number is required to produce the correct number of duplicates of a query.

Algorithm 3.3: COMPRESSAGGREGATED($(v1, v2, count, prev1, prev2)$)

comment: output: Header Byte, Deltav1, Deltav2, Count

```

if ( $v1 = prev1$ )
  then if ( $v2 - prev2 < 32 \wedge count < 5$ )
    then return ((( $count - 1$ ) * 32 + ( $v2 - prev2$ )), 0, 0, 0))
  else return (encode(0,  $v2 - prev2$ , count))
  else return (encode( $v1 - prev1$ ,  $v2$ , count))
  
```

Aggregated triple compression.

$v1=123$ $v2=24$ $count=30$ $prev1=123$ $prev2=20$
 Header=154 Deltav1=0 Deltav2=4 count=30

Gap	Payload	Delta	Delta	Delta
1	0011010	0	11	11110
Header		Deltav1	Deltav2	count

Aggregated Index Example.

RDF-3X-Physical Operators

IndexScan variables in a triple pattern are considered as a suffix in one of the six indices of the permutations of subject, predicate and object.

MergeJoin merges lexicographically sorted triples.

HashJoin a hash table with the left side of the join is created, triples from the right side are probed to find matches.

AggregatedIndexScan aggregated indices are traversed to recover the aggregated number of occurrences of the triple pattern prefix.

RDF-3X-Query Optimization Techniques

- Performs preliminary semantic analysis to check if the query produces empty results.
- A Dynamic programming based algorithm is implemented to generate left-linear plans whose leaves can be star-shaped sub-queries.
- An initial plan is generated as follows:

RDF-3X-Query Optimization Techniques

- Performs preliminary semantic analysis to check if the query produces empty results.
- A Dynamic programming based algorithm is implemented to generate left-linear plans whose leaves can be star-shaped sub-queries.
- An initial plan is generated as follows:
 - 1 Create an index scan for each triple pattern.
 - 2 Add a join between two triple patterns if they have a variable in common. If there is another join, add a selection.
 - 3 Add a selection containing all FILTER predicates.
 - 4 If the projection clause of a query includes the *distinct* option, add an aggregation operator that eliminates duplicates in the results.
 - 5 Add a dictionary lookup operator that converts the resulting ids back into strings.

RDF-3X-Query Optimization Techniques

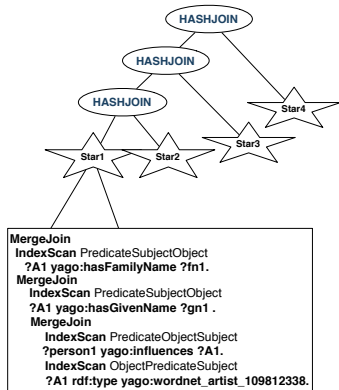
Parties of at most four artists that are influenced by at least one person:

```
SELECT ?A1 ?A2 ?A3 ?A4 WHERE
{ ?A1 yago:hasFamilyName ?fn1.
  ?A1 yago:hasGivenName ?gn1 .
  ?person1 yago:influences ?A1.
  ?A2 yago:hasFamilyName ?fn2 .
  ?A2 yago:hasGivenName ?gn2 .
  ?person1 yago:influences ?A2.
  ?A3 yago:hasFamilyName ?fn3.
  ?A3 yago:hasGivenName ?gn3 .
  ?person1 yago:influences ?A3.
  ?A4 yago:hasFamilyName ?fn4 .
  ?A4 yago:hasGivenName ?gn4 .
  ?person1 yago:influences ?A4.
  ?A1 rdf:type yago:wordnet_artist_109812338.
  ?A2 rdf:type yago:wordnet_artist_109812338.
  ?A3 rdf:type yago:wordnet_artist_109812338.
  ?A4 rdf:type yago:wordnet_artist_109812338.
```

RDF-3X-Query Optimization Techniques

Parties of at most four artists that are influenced by at least one person:

```
SELECT ?A1 ?A2 ?A3 ?A4 WHERE
{?A1 yago:hasFamilyName ?fn1.
 ?A1 yago:hasGivenName ?gn1 .
 ?person1 yago:influences ?A1.
 ?A2 yago:hasFamilyName ?fn2 .
 ?A2 yago:hasGivenName ?gn2 .
 ?person1 yago:influences ?A2.
 ?A3 yago:hasFamilyName ?fn3.
 ?A3 yago:hasGivenName ?gn3 .
 ?person1 yago:influences ?A3.
 ?A4 yago:hasFamilyName ?fn4 .
 ?A4 yago:hasGivenName ?gn4 .
 ?person1 yago:influences ?A4.
 ?A1 rdf:type yago:wordnet_artist_109812338.
 ?A2 rdf:type yago:wordnet_artist_109812338.
 ?A3 rdf:type yago:wordnet_artist_109812338.
 ?A4 rdf:type yago:wordnet_artist_109812338.
```



Left-linear plan of star-shape sub-queries.

Bit based Representations- BitMat. [6]

- 3D bit-cube representation where each dimension represents subjects, predicates and objects.
- Each cell of the matrix can be 0 or 1, and represents an RDF triple that can be formed by the combination of S, P, O.
- 3D bit-cube is sliced along a dimension to obtain 2D matrices.
- A gap compression schema on each bit-row is implemented:
- Basic operations over BitMat are defined to join basic triple patterns.

Bit based Representations- BitMat. [6]

- 3D bit-cube representation where each dimension represents subjects, predicates and objects.
- Each cell of the matrix can be 0 or 1, and represents an RDF triple that can be formed by the combination of S, P, O.
- 3D bit-cube is sliced along a dimension to obtain 2D matrices.
 - Inverting an S-O BitMat produces an O-S BitMat.
- A gap compression schema on each bit-row is implemented:
- Basic operations over BitMat are defined to join basic triple patterns.

Bit based Representations- BitMat. [6]

- 3D bit-cube representation where each dimension represents subjects, predicates and objects.
- Each cell of the matrix can be 0 or 1, and represents an RDF triple that can be formed by the combination of S, P, O.
- 3D bit-cube is sliced along a dimension to obtain 2D matrices.
 - Inverting an S-O BitMat produces an O-S BitMat.
- A gap compression schema on each bit-row is implemented:
 - A bit-row of "0011000" is represented as "[0]2 2 3".
- Basic operations over BitMat are defined to join basic triple patterns.

Bit based Representations- BitMat. [6]

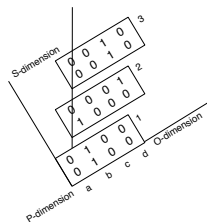
Subject	Property	Object
Wigle	sponsor	HR44
Rush	sponsor	HR44
Wigle	similarTo	Rush
Wigle	type	Person
Rush	type	Person
Rush	similarTo	Smith

Triples Table

Bit based Representations- BitMat. [6]

Subject	Property	Object
Wigle	sponsor	HR44
Rush	sponsor	HR44
Wigle	similarTo	Rush
Wigle	type	Person
Rush	type	Person
Rush	similarTo	Smith

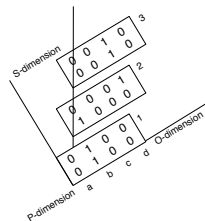
Triples Table



3D-cube BitMat

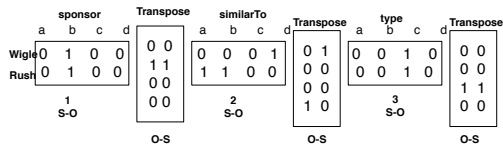
Bit based Representations- BitMat. [6]

Subject	Property	Object
Wigle	sponsor	HR44
Rush	sponsor	HR44
Wigle	similarTo	Rush
Wigle	type	Person
Rush	type	Person
Rush	similarTo	Smith



Triples Table

3D-cube BitMat



S-O and O-S BitMats for Ps

Hybrid Approach

- Support both native and DBMS-backed storages:

Hybrid Approach

- Support both native and DBMS-backed storages:
 - RedStore⁵ RDF triple store built on top of the Redland RDF library⁶; supports different native and DBMS-backed storage modules.
 - Sesame⁷, Java API that can be deployed as RDF triple storage or query services.
 - Virtuoso⁸ a universal data access middleware that integrates RDF with diverse data models: relational databases, Web data sources, files, etc.
 - BigData⁹ a high-performance RDF storage designed to scale to large datasets, and to support high concurrency and IO workloads on clusters and single machines.

⁵<http://www.aelius.com/njh/redstore/>

⁶<http://librdf.org/>

⁷<http://www.openrdf.org/documentation.jsp>

⁸<http://virtuoso.openlinksw.com/>

⁹<http://www.systap.com/bigdata.htm>

Virtuoso Sponger¹⁰

- Middleware hybrid storage solution able to integrate relational data, RDF, XML, and free text documents.
- URI/URL can be dereferenced within SPARQL query patterns:

¹⁰<http://virtuoso.openlinksw.com>

Virtuoso Sponger¹⁰

- Middleware hybrid storage solution able to integrate relational data, RDF, XML, and free text documents.
- URI/URL can be dereferenced within SPARQL query patterns:
 - Remote RDF data from a given URI, can be downloaded, parsed, and locally stored.
 - Triples from named graphs can be downloaded and stored.
 - Downloads and stores triples from remote RDF sources, based on a proximity search from a starting IRI to a given depth (# hops).
- Sponger functionalities are exposed by Virtuoso endpoints and can be specified via the SPARQL protocol.

¹⁰<http://virtuoso.openlinksw.com>

Sponging Settings-IRI dereferencing

Get Local Data Only considers only local data at the endpoint's dataset.

Get All Remote Data the SPARQL processor dereferences everything that might be useful for processing the query. Remote endpoints are contacted.

Get All Remote Data and Related Data *seeAlso* references in the dataset are used to find additional IRIs that will be dereferenced. This feature is required when triples needed to evaluate the query are accessible through remote endpoints.

Get Everything downloads all referred resources from remote endpoints.

Virtuoso-IRI dereferencing

```
SELECT ?id ?firstname ?nick
WHERE {
  GRAPH ?g {
    ?id rdf:type foaf:Person.
    ?id foaf:firstName ?firstname.
    ?id foaf:knows ?fn.
    ?fn foaf:nick ?nick.
    optional { ?id rdfs:SeeAlso ?more } } }
limit 10;
```

Virtuoso-IRI dereferencing

```
SELECT ?id ?firstname ?nick
WHERE {
  GRAPH ?g {
    ?id rdf:type foaf:Person.
    ?id foaf:firstName ?firstname.
    ?id foaf:knows ?fn.
    ?fn foaf:nick ?nick.
    optional {?id rdfs:SeeAlso ?more}}}
limit 10;
```

```
sparql
define input:grab-var "?more"
define input:grab-depth 10
define input:grab-limit 100
define input:grab-base "http://myhost/"
SELECT ?id ?firstname ?nick
WHERE {
  GRAPH ?g {
    ?id rdf:type foaf:Person.
    ?id foaf:firstName ?firstname.
    ?id foaf:knows ?fn.
    ?fn foaf:nick ?nick.
    optional {?id rdfs:SeeAlso ?more}}}
limit 10;
```

Virtuoso-IRI dereferencing

```
SELECT ?id ?firstname ?nick
WHERE {
  GRAPH ?g {
    ?id rdf:type foaf:Person.
    ?id foaf:firstName ?firstname.
    ?id foaf:knows ?fn.
    ?fn foaf:nick ?nick.
    optional { ?id rdfs:SeeAlso ?more } } }
limit 10;
```

```
sparql
define input:grab-var "?more"
define input:grab-depth 10
define input:grab-limit 100
define input:grab-base "http://myhost/"
SELECT ?id ?firstname ?nick
WHERE {
  GRAPH ?g {
    ?id rdf:type foaf:Person.
    ?id foaf:firstName ?firstname.
    ?id foaf:knows ?fn.
    ?fn foaf:nick ?nick.
    optional { ?id rdfs:SeeAlso ?more } } }
limit 10;
```

- input:grab-var** SPARQL variable identifying the IRI's to be downloaded.
- input:grab-depth** max # of links between the nodes in graph
- input:grab-limit** max # of (subject/object) nodes to retrieve
- input:grab-base** base IRI for converting relative IRIs to absolute IRIs plus others.

Virtuoso Endpoint ¹¹

Health care agencies that have conducted at least two different trials on the same drug. Number of answers 69.

```
SELECT DISTINCT ?A
```

```
WHERE {
```

```
?CL1 linkedct:intervention ?I.
```

```
?CL1 linkedct:source ?A.
```

```
?CL2 linkedct:source ?A.    []2in
```

```
?CL2 linkedct:intervention ?I.
```

```
?I rdf-schema:seeAlso ?DB.
```

```
?WK foaf:primaryTopic ?DB.
```

```
FILTER (?CL1 ≠ ?CL2)}
```

Sponge	# Answers
Local Data Only	NR
All Remote Data	NR
All Remote and Related Data	NR
Get Everything	NR

¹¹<http://hcls.deri.org/sparql>

Linked Data Query Processing

Top-Down Query Execution[17] based on a federated virtual integration of the Linked datasets.

- A mediator-wrapper architecture is implemented.
- A mediator discovers, retrieves and stores answers from the linked datasets.
- All source descriptions are available for query rewriting; state-of-the-art query rewriting techniques can be used to rewrite the query.
- A query plan is generated on the registered datasets based on available statistics.

Linked Data Query Processing

Bottom-Up Query Execution[19, 20, 21] no information is assumed to be available.

- Source discovery and ranking is performed on-the-fly.
- Scales up to large volume of unknown datasets.
- A query plan is generated on the registered datasets based on available statistics.

Linked Data Query Processing

Mixed Strategies[9, 27] combines both strategies by assuming some source descriptions are available , but new sources can be discovered during query processing.

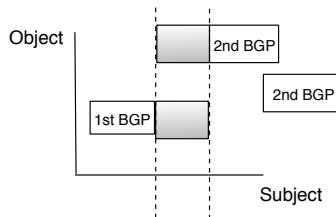
- Existing source descriptions can be used for query rewriting and optimization.
- During query execution, new sources are discovered and execution needs to be adapted to sources' availability.
- A query plan is generated on the registered datasets based on available statistics.

Top-Down Query Execution[17]

- A Qtree-based index structure stores data source descriptions and statistics collected in a pre-processing stage.
- A Qtree is a combination of histograms and an R-tree multidimensional structure.
- Information stored at the Qtree leaves are used for query rewriting and for source ranking.
- Regions in the Q-tree determine the best sources to answer a join query.

Example Top-Down Query Execution[17]

Identify the sources to implement the join between two BGP.



Overlap in the Qtree for joining Two BGPs.

Bottom-Up Query Execution[19, 20, 21]

Defines a link traversal based query execution on top of an extended iterator model; SQUIN¹² implements the proposed approach.

- A link traversal execution model is defined as a synchronized pipeline of operators that execute the iterators of the query in a fixed order.
- An execution is defined as a chain of iterators $I_1, \dots, I_n$.
- Each triple is associated with an iteration I_i .
- To execute the iterator I_i with the URI u_i , the iterator I_{i-1} had to completely dereference u_i .
- The reject operation is added to the iterator model.
- Heuristic-based optimization techniques are used to identify the order of the iterators that minimize the size of intermediate results.

¹²<http://squid.org>

Bottom-Up Query Execution[19, 20, 21]

Defines a link traversal based query execution on top of an extended iterator model; SQUIN¹² implements the proposed approach.

- A link traversal execution model is defined as a synchronized pipeline of operators that execute the iterators of the query in a fixed order.
- An execution is defined as a chain of iterators $I_1, \dots, I_n$.
- Each triple is associated with an iteration I_j .
- To execute the iterator I_j with the URI u_j , the iterator I_{j-1} had to completely dereference u_j .
- The reject operation is added to the iterator model.
 - Detects when a dereferenced dataset stops responding, to hide delays.
 - Submits other query requests to alive datasets.
- Heuristic-based optimization techniques are used to identify the order of the iterators that minimize the size of intermediate results.

¹²<http://squid.org>

Example Bottom-Up Query Execution[19, 20, 21]

```
SELECT ?P1 ?I1 ?P2 ?I2
WHERE {
  <http://olafhartig.de/foaf.rdf#olaf>
  <foaf:knows> ?P.
  ?P <foaf:Person> ?P1
  ?P1 <foaf:interest> ?I1.
  ?P1 <foaf:knows> ?P2.
  ?P2 <foaf:interest> ?I2}
```

Example Bottom-Up Query Execution[19, 20, 21]

```
SELECT ?P1 ?I1 ?P2 ?I2
WHERE {
  <http://olafhartig.de/foaf.rdf#olaf>
  <foaf:knows> ?P.
  ?P <foaf:Person> ?P1
  ?P1 <foaf:interest> ?I1.
  ?P1 <foaf:knows> ?P2.
  ?P2 <foaf:interest> ?I2}
```

```
I1 <http://olafhartig.de/foaf.rdf#olaf >
  <foaf:knows> ?P
I2 <http://page.mi.fu-berlin.de/luczak/foaf.rdf>
  <foaf:knows> ?P2
I3 <http://page.mi.fu-berlin.de/nixon/foaf.rdf>
  <foaf:knows> ?P2
I10 <http://www.informatik.uni-leipzig.de/ auer/foaf.rdf#me>
  <foaf:knows> ?P2
I30 <http://semanticweb.org/id/Sean.Bechhofer>
  <foaf:interest> ?I2
```

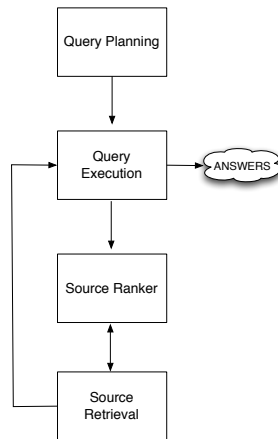
Iterators for triple patterns.

Mixed Strategies[27]

- Stream-based Linked Data Query Processing.
- Sources are selected using a Qtree that is updated when new sources are discovered.
- A query plan is built during query compilation based on the available sources.
- Each query operator is run in a different thread.
- New sources can be discovered during query execution time based on the content of the retrieved sources.
- Execution ends when all relevant sources have been processed.
- The non-blocking Symmetric Hash join is implemented to incrementally produce answers as soon as tuples become available.

Mixed Strategies[27]

Steps followed during Linked Data query processing.



Semantic Data Management[30]

A two-fold approach to optimize SPARQL queries over Linked Data.

- 1 During compilation time, analyzes the query and determines the classes of sources that do not contribute to the answer of the query.
- 2 During execution time, analyzes the query execution to discover more patterns which can be used to reduce the data to be retrieved, and the sources to be contacted.

Semantic Data Management[30]

A two-fold approach to optimize SPARQL queries over Linked Data.

- 1 During compilation time, analyzes the query and determines the classes of sources that do not contribute to the answer of the query.
- 2 During execution time, analyzes the query execution to discover more patterns which can be used to reduce the data to be retrieved, and the sources to be contacted.
 - A context graph is used by a heuristic to prune the space.

Example Semantic Data Management[30]

```
SELECT ?P1 ?P2
WHERE {
  <http://olafhartig.de/foaf.rdf#olaf>
  <foaf:knows> ?P.
  ?P <foaf:Person> ?P1
  ?P1 <yago:PhDadvisor> ?P2.
```

Context Graph is used to determine the instantiations of the variable ?P1 that belong to the class PhDStudent.

Summary

	Architecture	Sec. Memory Structures	Accessibility	Adaptivity	Semantic Data Management
RDF3X[29]	Native	Compressed Indices	Local	NO	Detect Empty Answers
Column-Store [1]	DBMS-backed	Vertical Partitions	Local	NO	YES
BitMat[6]	Native	3D Cube	Local	NO	NO
Virtuoso	Hybrid	RDF Quads Bitmap index	Remote Local	NO	YES
Top-Down Query Exec[17]	Native	Qtrees	Remote	NO	YES
Bottom-Up Query Exec[20, 21]	Hybrid		Remote	YES	YES
Mixed Query Exec[27]	Native		Remote	YES	YES
Semantic Data Man.[30]	Native		Remote	YES	YES

Lecture 3-The Adaptive Query Processing Paradigm

Limitations of the Optimize-Then-Execute Paradigm

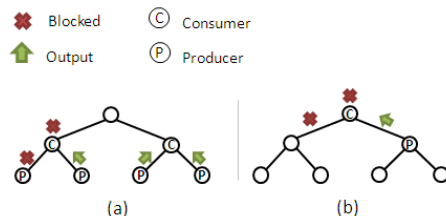
Query Processing is divided into three mayor steps:

- Statistics generation, done offline to collect information about the cardinalities, values frequencies, and costs to access data.
- Query optimization based on cost models, heuristic, and search strategies to identify "good-enough" plans. Assumptions:
 - Values of the variables
 - Availability of the data
 - Cardinality of operators
 - Behavior of data sources
- Query Execution which can be: pipelined, where each operator is processed at the time the output is produced by its sub-operators, or materialized, where operators are completely executed before firing its parents for processing.
- Good estimates for data with few correlations, executed in a relatively stable environment, and sufficient statistical information.

Limitations of the Optimize-Then-Execute Paradigm

Query Processing is divided into three major steps:

- Statistics generation, done offline to collect information about the cardinalities, values frequencies, and costs to access data.
- Query optimization based on cost models, heuristic, and search strategies to identify "good-enough" plans. Assumptions:
 - Values of the variables
 - Availability of the data
 - Cardinality of operators
 - Behavior of data sources
- Query Execution which can be: pipelined, where each operator is processed at the time the output is produced by its sub-operators, or materialized, where operators are completely executed before firing its parents for processing.
- Good estimates for data with few correlations, executed in a relatively stable environment, and sufficient statistical information.



Blocked Iterator Model.

Adaptive Query Processing [8, 13, 26]

Systems able to change their behavior by learning behavior of data providers.

- Receive information from the environment.
- Use up-to-date information to change their behavior.
- Keep iterating over time to adapt their behavior based on the environment conditions.

Adaptivity is needed:

- Misestimated or missing statistics.
- Unexpected correlations.
- Unpredictable costs.
- Dynamically changing data, workload and sources availability.
- Rates at which tuples arrive from the sources, change.

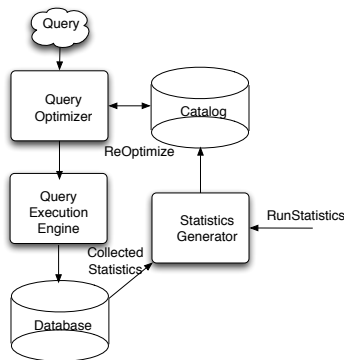
Initial Delays.

Slow Delivery

Bursty Arrivals

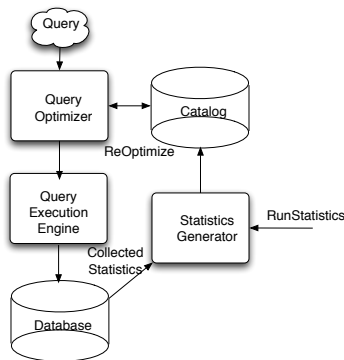
- Iterative environments.

Adaptive Query Processing

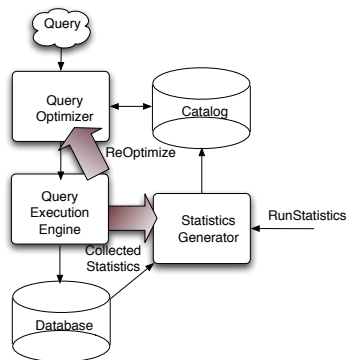


Typical Optimize-Then-Execute
Architecture.

Adaptive Query Processing



Typical Optimize-Then-Execute Architecture.



Adaptive Query Processing Architecture.

Evolution of Adaptive Query Processing Techniques

Early Relational Systems:

System R[32] keeps a catalog of statistics which can be updated; frequency of adaptivity is low.

Ingress[42] interleaves sub-plan generation and execution; for each query of n tables, selects the smallest table first, and for each tuple, the next-smallest table is probed for matches, and so on; the execution order can change from tuple to tuple.

Dynamic Query Plans[12] defines a set of constraints on changes of the environment, optimizes in advance and considers only a subset of all possible plans that respect these constraints; at runtime, results of the executed queries are used to tune the constraints.

Evolution of Adaptive Query Processing Techniques

- Adaptive Selectivity Estimation[11]** selects statistics about sizes of intermediate results at runtime, curve fitting is learned over time; statistics of non referenced tables are not gathered.
- DEC RDB[4]** multiple access methods are executed, and the most promising ones are chosen, while others are stopped.
- Mariposa[34]** requests bids for the cost of sub-plans, from each site; information about bids is stored, tuned each time the same sub-query is run, and used to identify the sources to run for each sub-plan.

Adaptive Query Processing Strategies

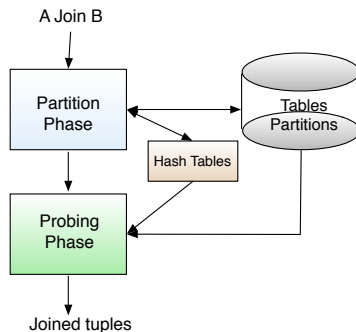
Adaptive Join Processing(Intra-Operator) adaptivity is performed at tuple level and query operators are able to adapt to the environment changes, even in the context of a fixed query plan.

Non-Pipelined Query Execution (Inter-Operator Reoptimization) sub-queries against remote data sources are re-scheduled based on uncertainties in the execution cost of the remote access, size of the materialized intermediate results, and unexpected delays.

Baseline Principles

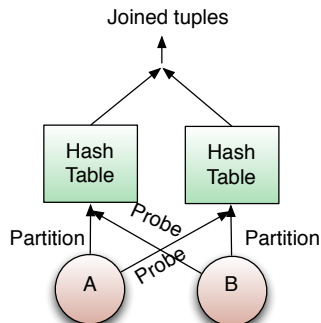
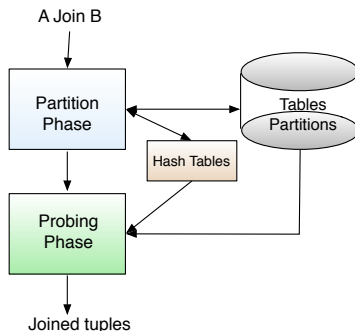
- Scheduler flexibility that allows adaptivity, but may consume more local memory and has execution overhead.
- Incrementally produce results as they become available.
- Processing progress even in presence of delays.

Symmetric Hash Join[13]



Traditional Hash Join blocks the probing phase until the partition phase is completely finished.

Symmetric Hash Join[13]



Traditional Hash Join blocks the probing phase until the partition phase is completely finished.

Symmetric Hash Join builds hash tables on both inputs; when a tuple arrives, it is inserted in its hash table and probed against the other table.

XJoin[36]

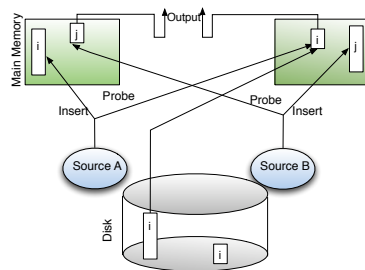
XJoin

- extends the Symmetric Hash Join to use secondary storage, overcoming the limitation that hash tables have to be stored in main memory until all of the tuples have been received.
- implements a reactively schedule background process which opportunistically utilizes delays to control background processing and produce tuples earlier.
- can be run for large inputs and in bushy trees.
- manages the data transfer between main and secondary memory.
- ensures completeness, and duplicates are not produced.
- works in three phases.

XJoin[36]

XJoin

- extends the Symmetric Hash Join to use secondary storage, overcoming the limitation that hash tables have to be stored in main memory until all of the tuples have been received.
- implements a reactive schedule background process which opportunistically utilizes delays to control background processing and produce tuples earlier.
- can be run for large inputs and in bushy trees.
- manages the data transfer between main and secondary memory.
- ensures completeness, and duplicates are not produced.
- works in three phases.



XJoin between tuples from Sources A and B.

XJoin-Phases

First Stage main memory partitioning and probing according to the join attribute values; matches are output; if main memory is full, one partition is chosen as victim and flushed to disk; it runs while at least one of the inputs produces tuples, or all the tuples have been received.

Second Stage activated when the first stage is blocked; one disk resident partition is picked, and tuples are probed against the corresponding partition of the other input in main memory; matches that correspond to non duplicate tuples are generated; the stage ends when one of the inputs starts producing tuples again.

Third Stage executes after all the tuples have been received, and makes sure that all the joined tuples are produced; it joins both main and secondary memory partitions.

XJoin-Avoiding Duplicates

Timestamp-based strategies are implemented in the second and third stages by maintaining a counter that registers if new tuples are received from a source or flushed into secondary memory:

- An arrival timestamp (ATS) is assigned to a tuple when it is received.
- A departure timestamp (DTS) is assigned to a tuple when it is flushed into disk.
- Two tuples are matched in the first stage, if they were in main memory in an overlapping interval.

XJoin-Avoiding Duplicates

Timestamp-based strategies are implemented in the second and third stages by maintaining a counter that registers if new tuples are received from a source or flushed into secondary memory:

- An arrival timestamp (ATS) is assigned to a tuple when it is received.
- A departure timestamp (DTS) is assigned to a tuple when it is flushed into disk.
- Two tuples are matched in the first stage, if they were in main memory in an overlapping interval.

	ATS	DTS		ATS	DTS		ATS	DTS
Tuple A1	200	400	Tuple B1	198	300	Tuple B2	401	500

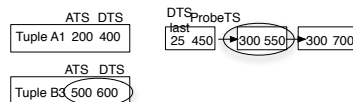
Tuples A1 and B1 were joined in the first stage, while A1 and B2 were not.

XJoin-Avoiding Duplicates

- A list of timestamps is assigned to each secondary memory partition, to represent the instant when the partition was accessed by the second stage(resp. third stage); entries are of the form $(DTS_{last}, ProbeTS)$ representing the DTS value of the last tuple of the disk-resident partition that was used to probe the main memory tuple, and the timestamp value when the second stage was executed, respectively.

XJoin-Avoiding Duplicates

- A list of timestamps is assigned to each secondary memory partition, to represent the instant when the partition was accessed by the second stage (resp. third stage); entries are of the form $(DTS_{last}, ProbeTS)$ representing the DTS value of the last tuple of the disk-resident partition that was used to probe the main memory tuple, and the timestamp value when the second stage was executed, respectively.



Tuples A1 and B3 were joined at instant 550 and should not be matched again.

MJoin

MJoin

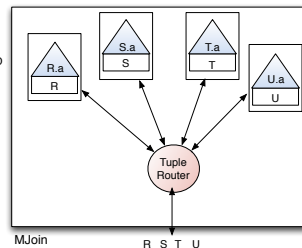
- generalizes the Symmetric Hash Join to multiple inputs; tuples can arrive in an arbitrary order.
- builds a hash index on every join attribute of every relation in the query.
- uses a tuple router to route it to the corresponding hash table.
- when a tuple arrives, first it is inserted in its corresponding hash table, and probed against the remaining hash tables to identify matches.
- performs very well if intermediate results are re-used, e.g., in star-shaped queries.
- The execution is history-independent because the internal state inside the hash tables is determined by the tuples' arrivals.

MJoin

MJoin

- generalizes the Symmetric Hash Join to multiple inputs; tuples can arrive in an arbitrary order.
- builds a hash index on every join attribute of every relation in the query.
- uses a tuple router to route it to the corresponding hash table.
- when a tuple arrives, first it is inserted in its corresponding hash table, and probed against the remaining hash tables to identify matches.
- performs very well if intermediate results are re-used, e.g., in star-shaped queries.
- The execution is history-independent because the internal state inside the hash tables is determined by the tuples' arrivals.

```
SELECT *
FROM R,S,T, U
WHERE R.a=S.a AND
      R.a=T.a AND
      R.a=U.a
```



MJoin between tuples from R, S, T and U.

The Eddy Routing Operator[7]

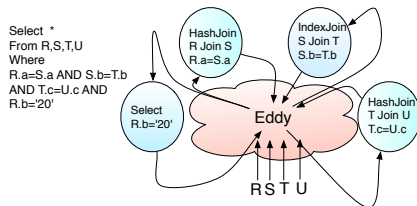
Eddy router operator:

- is an n -ary tuple router between n sources and a set of query operators.
- continuously reorders the execution of operators by routing each tuple through its operators in a variety of orders.
- its main objective is to ensure that tuples follow efficient routes, subject to the assumption that efficient routes can be found exploring different routes.
- each routing simulates a different pipelined plan; absence of an optimizer.
- merges multiple unary and binary operators into a single n -ary operator.
- operators' outputs are returned to the eddy.
- favors adaptivity over optimal performance.
- encapsulates the scheduling of its participating operators.
- operator selectivities and costs are maintained during execution.
- usually implemented as greedy solutions and designed for smaller, simpler plan spaces.

The Eddy Routing Operator[7]

Eddy router operator:

- is an n -ary tuple router between n sources and a set of query operators.
- continuously reorders the execution of operators by routing each tuple through its operators in a variety of orders.
- its main objective is to ensure that tuples follow efficient routes, subject to the assumption that efficient routes can be found exploring different routes.
- each routing simulates a different pipelined plan; absence of an optimizer.
- merges multiple unary and binary operators into a single n -ary operator.
- operators' outputs are returned to the eddy.
- favors adaptivity over optimal performance.
- encapsulates the scheduling of its participating operators.
- operator selectivities and costs are maintained during execution.
- usually implemented as greedy solutions and designed for smaller, simpler plan spaces.



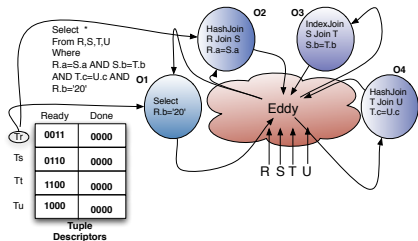
Eddy between tuples from R, S, T and U.

Eddy Operator

- Each tuple is described by a vector of Ready bits and Done bits, indicating operators eligible to process the tuple, and the ones that have already processed it.
- A tuple can be sent to one operator if the Ready bit for that operator is ON.
- Each tuple is sent to the output when all its Done bits are on.
- Done and Ready bits of triples output by operators, are set appropriately; Bits of tuples output by binary operators are ORed.
- Routing policies refer to the set of rules used by the eddy, to choose a routing destination for tuples among the possible valid destinations; they determine the efficiency of the operator.
- The Routing table stores the valid routing destinations for all the tuples.
- Statistics collected during execution can be used to tune the routing table.

Eddy Operator

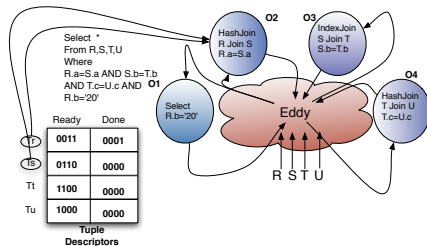
- Each tuple is described by a vector of Ready bits and Done bits, indicating operators eligible to process the tuple, and the ones that have already processed it.
- A tuple can be sent to one operator if the Ready bit for that operator is ON.
- Each tuple is sent to the output when all its Done bits are on.
- Done and Ready bits of triples output by operators, are set appropriately; Bits of tuples output by binary operators are OREd.
- Routing policies refer to the set of rules used by the eddy, to choose a routing destination for tuples among the possible valid destinations; they determine the efficiency of the operator.
- The Routing table stores the valid routing destinations for all the tuples.
- Statistics collected during execution can be used to tune the routing table.



Tr can be executed by O1 and O2.

Eddy Operator

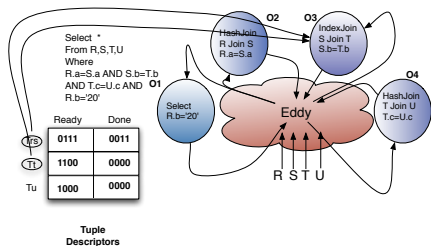
- Each tuple is described by a vector of Ready bits and Done bits, indicating operators eligible to process the tuple, and the ones that have already processed it.
- A tuple can be sent to one operator if the Ready bit for that operator is ON.
- Each tuple is sent to the output when all its Done bits are on.
- Done and Ready bits of triples output by operators, are set appropriately; Bits of tuples output by binary operators are ORed.
- Routing policies refer to the set of rules used by the eddy, to choose a routing destination for tuples among the possible valid destinations; they determine the efficiency of the operator.
- The Routing table stores the valid routing destinations for all the tuples.
- Statistics collected during execution can be used to tune the routing table.



Tuple descriptors after Tr was executed by O1.

Eddy Operator

- Each tuple is described by a vector of Ready bits and Done bits, indicating operators eligible to process the tuple, and the ones that have already processed it.
- A tuple can be sent to one operator if the Ready bit for that operator is ON.
- Each tuple is sent to the output when all its Done bits are on.
- Done and Ready bits of triples output by operators, are set appropriately; Bits of tuples output by binary operators are ORed.
- Routing policies refer to the set of rules used by the eddy, to choose a routing destination for tuples among the possible valid destinations; they determine the efficiency of the operator.
- The Routing table stores the valid routing destinations for all the tuples.
- Statistics collected during execution can be used to tune the routing table.



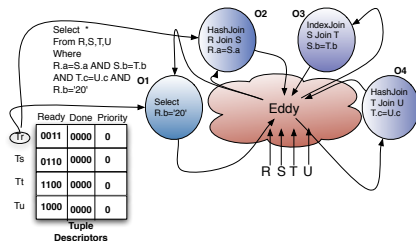
Tuple descriptors after Tr and Ts were executed by O2; new tuple Trs is added to the Routing Array.

Eddy Routing Policies-Priority Schema

- An eddy's tuple buffer is implemented as a priority queue with a flexible prioritization scheme.
- An operator is always given the highest-priority tuple in the buffer that has the corresponding Ready bit set.
- Tuples enter the eddy with low priority, and when they are returned to the eddy from an operator, they are given a higher priority.

Eddy Routing Policies-Priority Schema

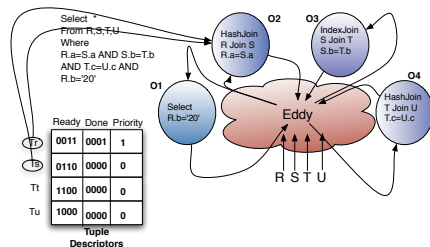
- An eddy's tuple buffer is implemented as a priority queue with a flexible prioritization scheme.
- An operator is always given the highest-priority tuple in the buffer that has the corresponding Ready bit set.
- Tuples enter the eddy with low priority, and when they are returned to the eddy from an operator, they are given a higher priority.



Initial priorities.

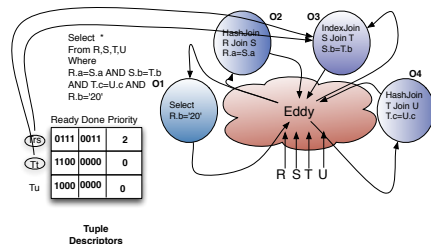
Eddy Routing Policies-Priority Schema

- An eddy's tuple buffer is implemented as a priority queue with a flexible prioritization scheme.
- An operator is always given the highest-priority tuple in the buffer that has the corresponding Ready bit set.
- Tuples enter the eddy with low priority, and when they are returned to the eddy from an operator, they are given a higher priority.



Eddy Routing Policies-Priority Schema

- An eddy's tuple buffer is implemented as a priority queue with a flexible prioritization scheme.
- An operator is always given the highest-priority tuple in the buffer that has the corresponding Ready bit set.
- Tuples enter the eddy with low priority, and when they are returned to the eddy from an operator, they are given a higher priority.



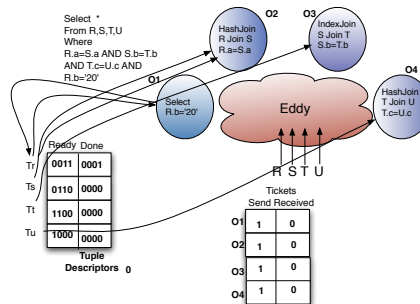
Trs's priority is the highest.

Eddy Routing Policies-Lottery Scheduling

- Tracks the relative efficiency of an operator at draining tuples from eddy.
- Maintains tickets for each operator.
- Each time eddy sends a tuple to an operator, it credits the operator with a ticket.
- A ticket is debited from the eddy's running count for an operator, when the operator returns a tuple.
- Each operator has an input queue.
- Holds a lottery among the operators eligible for receiving the tuple.
- Chances of receiving a tuple are proportional to the counts of tickets for that operator.
- The number of tickets for an operator is proportional to $1 - s$, where s is the selectivity of the operator.
- Statistics collected during query execution and the size of the operators' input queue, are used to tune the tickets.
- **Limitation:** Tickets do not always represent the behavior of join operators; if a join does not return tuples now, it may be able to return tuples later, when the matching tuples appear.

Eddy Routing Policies-Lottery Scheduling

- Tracks the relative efficiency of an operator at draining tuples from eddy.
- Maintains tickets for each operator.
- Each time eddy sends a tuple to an operator, it credits the operator with a ticket.
- A ticket is debited from the eddy's running count for an operator, when the operator returns a tuple.
- Each operator has an input queue.
- Holds a lottery among the operators eligible for receiving the tuple.
- Chances of receiving a tuple are proportional to the counts of tickets for that operator.
- The number of tickets for an operator is proportional to $1 - s$, where s is the selectivity of the operator.
- Statistics collected during query execution and the size of the operators' input queue, are used to tune the tickets.
- Limitation:** Tickets do not always represent the behavior of join operators; if a join does not return tuples now, it may be able to return tuples later, when the matching tuples appear.



Lottery and Tickets assignment.

Eddy Routing Policies-Content based Routing (CBR) [10]

- Explores and utilizes correlations between tuples' contents and operators' selectivity.
- Identifies the attribute tuple that is most strongly correlated with the selectivity of the operator, or *classifiers*.
- The probability of routing a tuple into an operator, is inversely proportional to its selectivity.
- Bayesian network based approaches can be used to identify conditional dependencies and learn the classifiers.

Eddy Routing Policies-Content based Routing (CBR) [10]

- Explores and utilizes correlations between tuples' contents and operators' selectivity.
- Identifies the attribute tuple that is most strongly correlated with the selectivity of the operator, or *classifiers*.
- The probability of routing a tuple into an operator, is inversely proportional to its selectivity.
- Bayesian network based approaches can be used to identify conditional dependencies and learn the classifiers.

Values of A	O1	O2	O3
A="a"	40%	30%	50%
A="b"	38%	40%	20%
A="c"	10%	60%	80%
Overall	29%	46%	50%

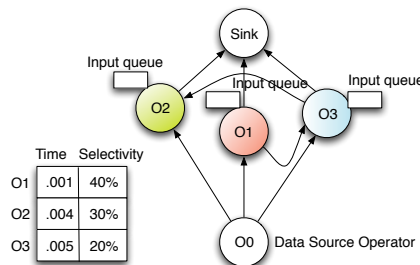
- Best overall orders, O1, O2, and O3.
- Best order for tuples with A="a": O2, O1, and O3.
- Best order for tuples with A="b": O3, O1, and O2.
- Best order for tuples with A="c": O1, O2, and O3.

Distributed Eddies[35]

- Centralized Eddies may incur into unnecessary network traffic and delays when operators are remotely executed.
- To avoid communication with a centralized eddy, operators can make routing decisions in a distributed fashion.
- A set of operators $O_1, \dots, O_n$ are connected by a network, and each one generates tuples.
- Each tuple is annotated with the timestamp when it was generated, and its own execution history, i.e., the operators that the tuple has visited so far; operators that the tuple still has to visit are determined by the operators that have already been visited.
- For each output tuple t , an operator O_i selects the next operator to send t with probability proportional to a routing weight function $w_1(h), \dots, w_n(h)$ where n is the number of operators in the distributed plan.
- Tuples are passed through the eligible operators, until the Data Sink operator is reached.
- Sink operators compute the response time of the tuple in the system, and the sequence of operators visited by the tuple, corresponds to the execution plan of the tuple.

Distributed Eddies[35]

- Centralized Eddies may incur into unnecessary network traffic and delays when operators are remotely executed.
- To avoid communication with a centralized eddy, operators can make routing decisions in a distributed fashion.
- A set of operators $O_1, \dots, O_n$ are connected by a network, and each one generates tuples.
- Each tuple is annotated with the timestamp when it was generated, and its own execution history, i.e., the operators that the tuple has visited so far; operators that the tuple still has to visit are determined by the operators that have already been visited.
- For each output tuple t , an operator O_i selects the next operator to send t with probability proportional to a routing weight function $w_1(h), \dots, w_n(h)$ where n is the number of operators in the distributed plan.
- Tuples are passed through the eligible operators, until the Data Sink operator is reached.
- Sink operators compute the response time of the tuple in the system, and the sequence of operators visited by the tuple, corresponds to the execution plan of the tuple.



Each operator has a fixed number of resources, and tuples should be routed to operators with lower service times and more selective.

Routing Policies

- Back-Pressure** more heavily loaded operators will force tuples to be routed to more lightly loaded operators. Tuples are routed to operators with shortest queue length.
- Smart Eddy** tickets are used as indicators of learned selectivity of each operator; tuples are routed to operators with largest tickets, i.e., most selective. Operators turn their tickets to 0 when there are overloaded.
- Selectivity Cost** tuples are routed to the next eligible operator with the biggest benefit, where the benefit represents the possibility of eliminating a tuple in an operator divided by the cost of processing the tuple.

Inter-Operator Re-optimization[8, 25, 28, 37]

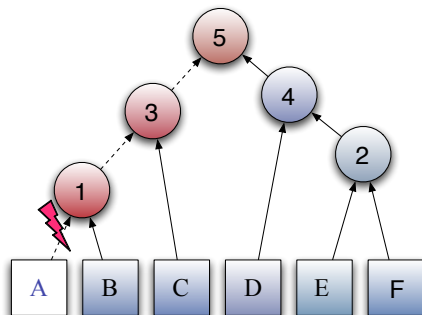
- The optimize-then-execute paradigm is extended to monitoring plans' executions and re-optimizing whenever when the environment conditions differ from the ones considered by the optimizer.
- Lack of statistics and system conditions' information.
- Detect and correct situations where optimizers select sub-optimal plans, which could be improved if accurate statistics are considered.
- Existing approaches extend statistics tracking with a log of estimates considered by the optimizer to generate the query plan, and of system conditions during query execution.
- Physical operators are implemented to track statistics and fire re-optimizations.
- Re-optimization must ensure that a new plan will not duplicate or miss outputs, and reuse previous computations.
- Should avoid thrashing, i.e., spend most of the time and resources, monitoring operators.

Query Scrambling[37, 38]

- A reactive solution that first generates a plan, and during query execution reacts to misestimations.
- Query Scrambling is an execution query method that reacts to data arrival delays.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable subtrees are identified, i.e., subtree where all the operators are runnable.
- Executed sub-trees are materialized until the delayed sources become available and the original plan can resumed.
- Query optimizer is invoked to identify non-delayed sub-trees that can be materialized first.
- New operators can be introduced to join tables that were not directly joined in the original plan.

Query Scrambling[37, 38]

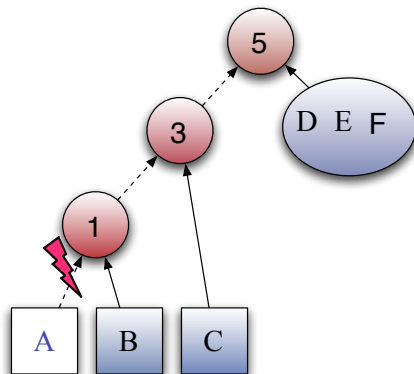
- A reactive solution that first generates a plan, and during query execution reacts to misestimations.
- Query Scrambling is an execution query method that reacts to data arrival delays.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable subtrees are identified, i.e., subtree where all the operators are runnable.
- Executed sub-trees are materialized until the delayed sources become available and the original plan can be resumed.
- Query optimizer is invoked to identify non-delayed sub-trees that can be materialized first.
- New operators can be introduced to join tables that were not directly joined in the original plan.



Source A is delayed.

Query Scrambling[37, 38]

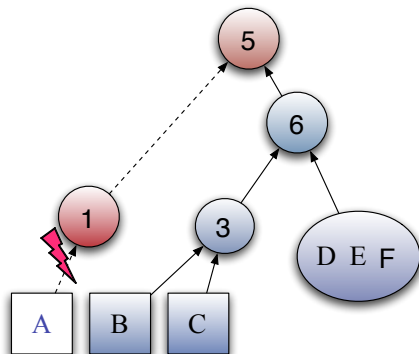
- A reactive solution that first generates a plan, and during query execution reacts to misestimations.
- Query Scrambling is an execution query method that reacts to data arrival delays.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable subtrees are identified, i.e., subtree where all the operators are runnable.
- Executed sub-trees are materialized until the delayed sources become available and the original plan can be resumed.
- Query optimizer is invoked to identify non-delayed sub-trees that can be materialized first.
- New operators can be introduced to join tables that were not directly joined in the original plan.



Sub-tree 4 is materialized.

Query Scrambling[37, 38]

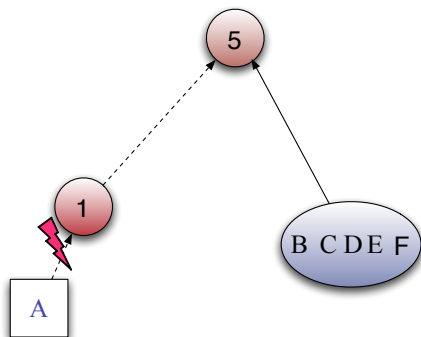
- A reactive solution that first generates a plan, and during query execution reacts to misestimations.
- Query Scrambling is an execution query method that reacts to data arrival delays.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable subtrees are identified, i.e., subtree where all the operators are runnable.
- Executed sub-trees are materialized until the delayed sources become available and the original plan can be resumed.
- Query optimizer is invoked to identify non-delayed sub-trees that can be materialized first.
- New operators can be introduced to join tables that were not directly joined in the original plan.



A new operator is created between node 3 and D, E and F.

Query Scrambling[37, 38]

- A reactive solution that first generates a plan, and during query execution reacts to misestimations.
- Query Scrambling is an execution query method that reacts to data arrival delays.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable subtrees are identified, i.e., subtree where all the operators are runnable.
- Executed sub-trees are materialized until the delayed sources become available and the original plan can be resumed.
- Query optimizer is invoked to identify non-delayed sub-trees that can be materialized first.
- New operators can be introduced to join tables that were not directly joined in the original plan.



The new operator is executed.

Summary Database Adaptive Query Processing Techniques

- Adaptive Query Processing Techniques can be at the operator or plan level.
- Intra-operator techniques provide physical operators able to access remote data, hide data sources' delays, and adapt their execution to data availability.
- Inter-operator strategies extend classical plan-first-execute-then paradigm with the ability to monitoring plans' executions and re-optimize sub-optimal plans.
- Routing techniques avoid traditional optimizers, and generate plans for each tuple during runtime by routing tuples through the pool of eligible operators.
- Routing policies are implemented to guide tuples into efficient routes and detecting tuples that will not belong to the answer sooner.
- Effectiveness of adaptive query processing techniques depends on accurate of statistics about selectivity, execution cost, and plan benefits; so techniques to learn this knowledge are required.

Lecture 4: An Adaptive Paradigm for Linked Data

SPARQL Endpoints

- Implement SPARQL protocol and enable users to query particular datasets or to dereference linked data.
- Developed for very light-weight use; so they very likely timeout without producing any answer for complex queries.
- SPARQL SPONGE parameter can be used to dereference URI's:

Local: considers only local data at the endpoint's dataset.

Grab All: the SPARQL processor dereferences everything that might be useful for processing the query. Remote endpoints are contacted.

Grab All (seeAlso): *seeAlso* references in the dataset are used to find additional IRIs that will be dereferenced. This feature is required when triples needed to evaluate the query are accessible through remote endpoints.

Grab Everything: downloads all referred resources from remote endpoints.



SPARQL Endpoints' Limitations



- *Executions* against several endpoints can be *unsuccessful* because of endpoint timeouts, unpredictable data transfers and endpoint availability.
- Queries need to be *decomposed* into simple *sub-queries* that the endpoint will be able to successfully execute before timed out.
- Operators to gather information from different endpoints need to be able to *incrementally* produce results, and *re-optimize* queries whenever an endpoint unpredictably becomes *blocked*.
- Requests have to be *routed* through eligible endpoints.
- Existing RDF triple stores have developed strategies to efficiently store and access RDF data *locally*, but many of them are *not* able to gather data accessible through SPARQL endpoints, or *hide* delays from users.

Motivating Example

Sensors that detected freezing temperatures on April 1, 2003, between 1:00- 3:00am.

```
PREFIX om-owl:<http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weather:<http://knoesis.wright.edu/ssw/ont/weather.owl#>
PREFIX sens-obs:<http://knoesis.wright.edu/ssw/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX owl-time:<http://www.w3.org/2006/time#>
PREFIX gn:<http://www.geonames.org/ontology#>
SELECT DISTINCT *
WHERE {
  ?sensor om-owl:generatedObservation ?observation .
  ?observation rdf:type weather:TemperatureObservation .
  ?observation om-owl:samplingTime ?time .?time owl-time:inXSDDateTime ?xsdtime .
  ?observation om-owl:result ?result .?result om-owl:floatValue ?value .
  ?result om-owl:uom weather:fahrenheit .FILTER(?value <= "32.0" xsd:float).
  FILTER(?xsdtime >= "2003-04-01T01:00:00-07:00 http://www.w3.org/2001/XMLSchema#dateTime").
  FILTER(?xsdtime <= "2003-04-01T03:00:00-07:00 http://www.w3.org/2001/XMLSchema#dateTime").
  ?sensor om-owl:hasLocatedNearRel ?location .?location om-owl:hasLocation ?ga. ?ga gn:name ?name}
```

Motivating Example

Sensors that detected freezing temperatures on April 1, 2003, between 1:00- 3:00am.

```

PREFIX om-owl:<http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weather:<http://knoesis.wright.edu/ssw/ont/weather.owl#>
PREFIX sens-obs:<http://knoesis.wright.edu/ssw/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX owl-time:<http://www.w3.org/2006/time#>
PREFIX gn:<http://www.geonames.org/ontology#>
SELECT DISTINCT *
WHERE {
  ?sensor om-owl:generatedObservation ?observation .
  ?observation rdf:type weather:TemperatureObservation .
  ?observation om-owl:samplingTime ?time .?time owl-time:inXSDDateTime ?xsdtime .
  ?observation om-owl:result ?result .?result om-owl:floatValue ?value .
  ?result om-owl:uom weather:fahrenheit .FILTER(?value <= " 32.0" xsd:float).
  FILTER(?xsdtime >= "2003-04-01T01:00:00-07:00 http://www.w3.org/2001/XMLSchema#dateTime").
  FILTER(?xsdtime <= "2003-04-01T03:00:00-07:00 http://www.w3.org/2001/XMLSchema#dateTime").
  ?sensor om-owl:hasLocatedNearRel ?location .?location om-owl:hasLocation ?ga. ?ga gn:name ?name}

```

	Local	Grab All	Grab All (<i>seeAlso</i>)	Grab Everthing
Average	0.3500	100.7864	Timeout	Timeout
Standard Deviation	0.0472	38.3228	Timeout	Timeout
Minimum	0.3000	58.9500	Timeout	Timeout
Maximum	0.4500	155.5900	Timeout	Timeout

Motivating Example

Sensors that detected freezing temperatures on April 1, 2003, between 1:00- 3:00am.

```

PREFIX om-owl:<http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX weather:<http://knoesis.wright.edu/ssw/ont/weather.owl#>
PREFIX sens-obs:<http://knoesis.wright.edu/ssw/>
PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
PREFIX owl-time:<http://www.w3.org/2006/time#>
PREFIX gn:<http://www.geonames.org/ontology#>
SELECT DISTINCT *
WHERE {
  ?sensor om-owl:generatedObservation ?observation .
  ?observation rdf:type weather:TemperatureObservation .
  ?observation om-owl:samplingTime ?time .?time owl-time:inXSDDateTime ?xsdtime .
  ?observation om-owl:result ?result .?result om-owl:floatValue ?value .
  ?result om-owl:uom weather:fahrenheit .FILTER(?value <= " 32.0" xsd:float).
  FILTER(?xsdtime >= "2003-04-01T01:00:00-07:00 http://www.w3.org/2001/XMLSchema#dateTime").
  FILTER(?xsdtime <= "2003-04-01T03:00:00-07:00 http://www.w3.org/2001/XMLSchema#dateTime").
  ?sensor om-owl:hasLocatedNearRel ?location .?location om-owl:hasLocation ?ga. ?ga gn:name ?name}

```

	Local	Grab All	Grab All (seeAlso)	Grab Everything
Average	0.3500	100.7864	Timeout	Timeout
Standard Deviation	0.0472	38.3228	Timeout	Timeout
Minimum	0.3000	58.9500	Timeout	Timeout
Maximum	0.4500	155.5900	Timeout	Timeout

LinkedSensorData endpoint may become blocked waiting for answers from other endpoints.

Motivating Examples

Clinical trials and drugs used for Breast, Colorectal, Ovarian, and Lung Cancer.

```

PREFIX linkedct: <http://data.linkedct.org/resource/linkedct/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX dbpedia: <http://dbpedia.org/property/>
PREFIX elements: <http://purl.org/dc/elements/1.1/>
SELECT DISTINCT ?fn1 ?fn2 ?fn3 ?fn4 ?I ?DB ?ID1 ?ID2 ?L
WHERE {
  ?A1 foaf:page ?fn1. ?A1 linkedct:condition ?A2.
  ?A2 linkedct:condition_name "Colorectal Cancer".
  ?A1 linkedct:intervention ?I.
  ?A3 foaf:page ?fn3. ?A3 linkedct:condition ?A4.
  ?A4 linkedct:condition_name "Breast Cancer".
  ?A3 linkedct:intervention ?I.
  ?A5 foaf:page ?fn5. ?A5 linkedct:condition ?A6.
  ?A6 linkedct:condition_name "Ovarian Cancer".
  ?A5 linkedct:intervention ?I.
  ?A7 foaf:page ?fn7. ?A7 linkedct:condition ?A8.
  ?A8 linkedct:condition_name "Lung Cancer".
  ?A7 linkedct:intervention ?I
  ?I linkedct:intervention_type "Drug".
  ?I rdf:seeAlso ?DB. ?WK foaf:primaryTopic ?DB.
  ?WK dbpedia:revisionId ?ID1.
  ?WK dbpedia:pageId ?ID2. ?WK elements:language ?L.}

```

Lecture 4: An Adaptive Paradigm for Linked Data

Requirements for Adaptive Semantic Data Management Techniques for Linked Data

Motivating Examples

Clinical trials and drugs used for Breast, Colorectal, Ovarian, and Lung Cancer.

PREFIX linkedct: <http://data.linkedct.org/resource/linkedct />

PREFIX foaf:<http://xmlns.com/foaf/0.1/>

PREFIX rdf:<http://www.w3.org/2000/01/rdf-schema#>

PREFIX dbpedia:<http://dbpedia.org/property/>

PREFIX elements:<http://purl.org/dc/elements/1.1/>

SELECT DISTINCT ?fn1 ?fn2 ?fn3 ?fn4 ?l ?DB ?ID1 ?ID2 ?L

WHERE {

?A1 foaf:page ?fn1.?A1 linkedct:condition ?A2.

?A2 linkedct:condition_name "Colorectal Cancer".

?A1 linkedct:intervention ?l.

?A3 foaf:page ?fn3.?A3 linkedct:condition ?A4.

?A4 linkedct:condition_name "Breast Cancer".

?A3 linkedct:intervention ?l.

?A5 foaf:page ?fn5.?A5 linkedct:condition ?A6.

?A6 linkedct:condition_name "Ovarian Cancer".

?A5 linkedct:intervention ?l.

?A7 foaf:page?fn7.?A7 linkedct:condition ?A8.

?A8 linkedct:condition_name "Lung Cancer".

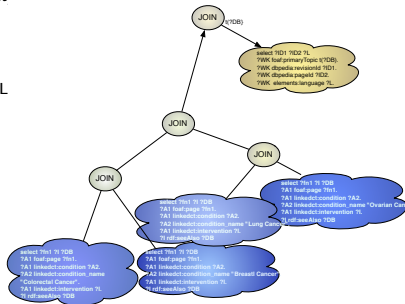
?A7 linkedct:intervention ?l

?l linkedct:intervention_type "Drug".

?l rdf:seeAlso ?DB. ?WK foaf:primaryTopic ?DB.

?WK dbpedia:revisionId ?ID1.

?WK dbpedia:pageId ?ID2.?WK elements:language ?L.}



A star-shaped plan; four star-shaped sub-queries against LinkedCT endpoints and one against DBpedia endpoint.

Motivating Examples

Clinical trials and drugs used for Breast, Colorectal, Ovarian, and Lung Cancer.

PREFIX linkedct: <http://data.linkedct.org/resource/linkedct />

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

PREFIX rdf: <http://www.w3.org/2000/01/rdf-schema#>

PREFIX dbpedia: <http://dbpedia.org/property/>

PREFIX elements: <http://purl.org/dc/elements/1.1/>

SELECT DISTINCT ?fn1 ?fn2 ?fn3 ?fn4 ?I ?DB ?ID1 ?ID2 ?L

WHERE {

?A1 foaf:page ?fn1. ?A1 linkedct:condition ?A2.

?A2 linkedct:condition_name "Colorectal Cancer".

?A1 linkedct:intervention ?I.

?A3 foaf:page ?fn3. ?A3 linkedct:condition ?A4.

?A4 linkedct:condition_name "Breast Cancer".

?A3 linkedct:intervention ?I.

?A5 foaf:page ?fn5. ?A5 linkedct:condition ?A6.

?A6 linkedct:condition_name "Ovarian Cancer".

?A5 linkedct:intervention ?I.

?A7 foaf:page ?fn7. ?A7 linkedct:condition ?A8.

?A8 linkedct:condition_name "Lung Cancer".

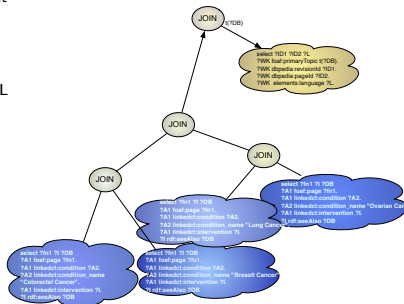
?A7 linkedct:intervention ?I

?I linkedct:intervention_type "Drug".

?I rdf:seeAlso ?DB. ?WK foaf:primaryTopic ?DB.

?WK dbpedia:revisionId ?ID1.

?WK dbpedia:pageId ?ID2. ?WK elements:language ?L. }



A star-shaped plan; four star-shaped sub-queries against LinkedCT endpoints and one against DBpedia endpoint.

Delays should be hidden and semantics of the data should be used at runtime to reduce unnecessary calls to endpoints.

Existing Approaches-Adaptivity Capacity

Routing Strategies:

- Hartig et al.[19, 20, 21] implement an eddy based solution where iterators correspond to operators and routing policy is guided by four rules.

Intra-Operators based Solutions:

- Ladwig and Tran[27] implement a non-blocking operator for Linked Data.

Existing Approaches-Adaptivity Capacity

Routing Strategies:

- Hartig et al.[19, 20, 21] implement an eddy based solution where iterators correspond to operators and routing policy is guided by four rules.

Intra-Operators based Solutions:

- Ladwig and Tran[27] implement a non-blocking operator for Linked Data.

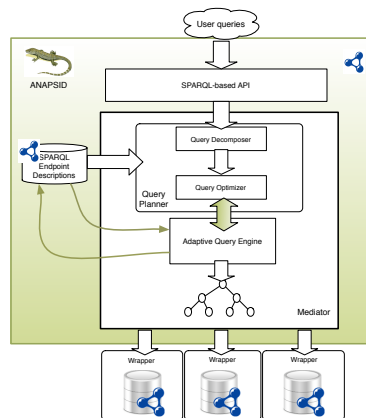
What is needed?

Exploit semantics of the endpoints' data as well as quality of endpoints parameters in:

- Routing strategies.
- Intra- and Inter-operator techniques.
- Correlation estimation techniques.
- Ranking strategies.

ANAPSID: An Adaptive Query Processing Engine for Linked Data[3]

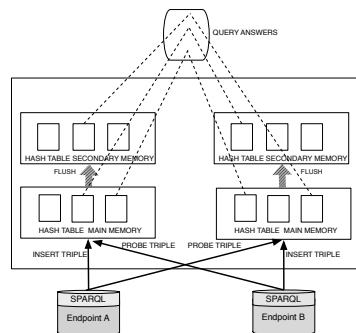
- Engine for SPARQL endpoints that extends adaptive query processing to exploit knowledge encoded in the Linked dataset ontologies, and to access Linked Data through SPARQL endpoints.
- Based on the architecture of wrappers and mediators.
- Wrappers are created around SPARQL endpoints to properly build the URL to contact the endpoint, and to convert the results in appropriate formats.
- ANAPSID decompose SPARQL queries into sub-queries against the endpoints[5, 24].
- Mediators maintain information about endpoint capabilities and statistics that describe their content and performance, to decompose user queries into sub-queries in the available endpoints.
- ANAPSID implements a query engine which provides a set of operators able to gather data from different endpoints, and opportunistically produce results by joining triples even when one of the endpoints becomes blocked.



Intra-Operator Techniques

Adaptive Group Join (agjoin):

- Extends the Xjoin[36] and gjoin[39] operators; quickly produces answers.
- Uses the distribution of values in the different RDF documents, to decide the number and size of the buckets of each of the hash tables.
- RDF data distributions are used to decide the bucket or partition that will be flushed to secondary memory when a partition is full.

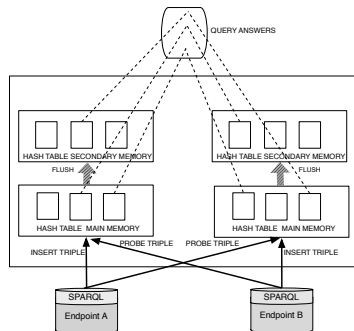


Adaptive Group Join (agjoin)

Intra-Operator Techniques

Adaptive Group Join (agjoin):

- Extends the Xjoin[36] and gjoin[39] operators; quickly produces answers.
- Uses the distribution of values in the different RDF documents, to decide the number and size of the buckets of each of the hash tables.
- RDF data distributions are used to decide the bucket or partition that will be flushed to secondary memory when a partition is full.



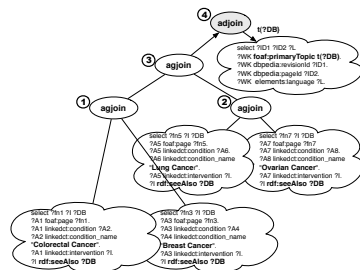
Adaptive Group Join (agjoin)

Additional Operators:

- **ADjoin** extends the Dependent join operator [14, 43] with the capability to hide delays to the user and use knowledge encoded in the ontology to reduce unnecessary endpoints' access; required when a certain binding pattern has to be respected during the execution of a query.
- **AUnion** adaptive versions of the *UNION*.
- **AOptional** adaptive versions of the *OPTIONAL*.

Inter-Operator Techniques

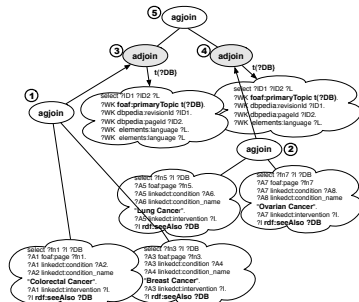
- ANAPSID implements the delay-hiding feature *Query Scrambling* [38].
- Original plans are re-optimized when any of the operators stops producing results; new plan built on-the-fly based on endpoints' availability.
- New operators can be added to plan to execute the query in small stages or to produce partial answers; e.g., it may change an **adjoin** by an **optional**, and treat joins with unavailable triples as non matching joins.



Original Plan **adjoin** 2 stops producing results because its producers are blocked.

Inter-Operator Techniques

- ANAPSID implements the delay-hiding feature *Query Scrambling* [38].
- Original plans are re-optimized when any of the operators stops producing results; new plan built on-the-fly based on endpoints' availability.
- New operators can be added to plan to execute the query in small stages or to produce partial answers; e.g., it may change an **adjoin** by an **optional**, and treat joins with unavailable triples as non matching joins.



Adaptive Query Engine-Re-Optimization; **adjoin** is distributed over the **agjoins**.

Initial Experimental Study

- **Datasets** *LinkedSensorData-blizzards*^a and *linkedCT*^b; Sensor data ^c is a Virtuoso SPARQL endpoints. The timeout for the Virtuoso endpoint <http://sonicbanana.cs.wright.edu:8890/sparql> was changed to 86,400 secs. (one day).
- **Query Benchmarks** Benchmark 1 is a set of 10 queries against *LinkedSensorData-blizzards* comprised of between 24 and 30 triple patterns; they can be grouped into 4 or 5 local sub-queries against the same SPARQL endpoint. Benchmark 2 is a set of 10 queries over *linkedCT*; queries are composed of between 13 and 17 patterns that can be grouped into 3 or 4 local sub-queries against the *linkedCT* SPARQL endpoint.

Dataset	Number of triples
LinkedSensorData-blizzards	56,689,107
linkedCT	9,809,330

^a<http://wiki.knoesis.org/index.php/LinkedSensorData>

^blinkedCT.org

^c<http://sonicbanana.cs.wright.edu:8890/sparql>

Experimental Results

Benchmark 1									
SPARQL Endpoint	q1	q2	q3	q4	q5	q6	q7	q8	Geom. Mean
	AVG 380.71	AVG 147.03	AVG 141.06	AVG 93.86	AVG 464.02	AVG 330.16	AVG 466.62	AVG 198.86	AVG 234.86
	STD 63.73	STD 3.68	STD 1.89	STD 2.63	STD 18.25	STD 5.63	STD 12.30	STD 2.02	STD 7.33
ANAPSID	AVG 16.60	AVG 9.54	AVG 6.80	AVG 9.59	AVG 14.34	AVG 13.48	AVG 11.08	AVG 16.19	AVG 12.16
	STD 1.14	STD 1.06	STD 0.30	STD 0.26	STD 2.90	STD 1.98	STD 0.08	STD 1.65	STD 0.68
Benchmark 2									
SPARQL Endpoint	q1	q2	q3	q4	q5	q6	q7	q8	Geom. Mean
	-	-	-	-	-	-	-	-	-
ANAPSID	AVG 6.21	AVG 6.11	AVG 6.67	AVG 6.24	AVG 6.89	AVG 6.76	AVG 4.28	AVG 1.10	AVG 5.30
	STD 1.04	STD 0.31	STD 0.98	STD 0.58	STD 0.93	STD 0.31	STD 0.91	STD 0.30	STD 0.30

Benchmark 1 ANAPSID improves SPARQL endpoint execution time by a factor of 19.31.

Benchmark 2 all queries timed out in all linkedCT SPARQL endpoints: server stopped responding and proxy error 502.

Future Directions

- Eddy operators and routing strategies have to be defined to exploit semantics and quality of endpoints parameters.
- Intra- and Inter-operator techniques should also consider semantics and quality information.
- Techniques to learn dependencies between data.
- Ranking strategies.
- Query Rewriting Techniques.

- [1] Daniel J. Abadi, Adam Marcus, Samuel Madden, and Kate Hollenbach. SW-Store: a vertically partitioned DBMS for Semantic Web data management. *VLDB J.*, 18(2):385–406, 2009.
- [2] Daniel J. Abadi, Adam Marcus, Samuel Madden, and Katherine J. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *VLDB*, pages 411–422, 2007.
- [3] Maribel Acosta, Maria-Esther Vidal, Edna Ruckhaus, Julio Castillo, and Tomas Lampo. ANAPSID: AN Adaptive query ProcesSing engine for sparql enDpoints. In *ESWC-Poster*, 2011.
- [4] Gennady Antoshenkov and Mohamed Ziauddin. Query processing and optimization in oracle rdb. *VLDB J.*, 5(4):229–237, 1996.

- [5] Yolifé Arvelo, Blai Bonet, and Maria-Esther Vidal. Compilation of query-rewriting problems into tractable fragments of propositional logic. In *AAAI*, 2006.
- [6] Medha Atre, Vineet Chaoji, Mohammed J. Zaki, and James A. Hendler. Matrix "Bit" loaded: a scalable lightweight join query processor for RDF data. In *Proceedings of the WWW*, pages 41–50, 2010.
- [7] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD Conference*, pages 261–272, 2000.
- [8] Shivnath Babu and Pedro Bizarro. Adaptive query processing in the looking glass. In *CIDR*, pages 238–249, 2005.
- [9] Cosmin Basca and Abraham Bernstein. Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In *The 6th International Workshop on Scalable Semantic Web Knowledge Base Systems at ISWC*, 2010.

- [10] Pedro Bizarro, Shivnath Babu, David J. DeWitt, and Jennifer Widom. Content-based routing: Different plans for different data. In *VLDB*, pages 757–768, 2005.
- [11] Chung-Min Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In *SIGMOD Conference*, pages 161–172, 1994.
- [12] Richard L. Cole and Goetz Graefe. Optimization of dynamic query evaluation plans. In *SIGMOD Conference*, pages 150–160, 1994.
- [13] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
- [14] Daniela Florescu, Alon Y. Levy, Ioana Manolescu, and Dan Suciu. Query optimization in the presence of limited access patterns. In *SIGMOD Conference*, pages 311–322, 1999.

- [15] Lise Getoor, Benjamin Taskar, and Daphne Koller. Selectivity estimation using probabilistic models. In *SIGMOD Conference*, pages 461–472, 2001.
- [16] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.
- [17] Andreas Harth, Katja Hose, Marcel Karnstedt, Axel Polleres, Kai-Uwe Sattler, and Jürgen Umbrich. Data summaries for on-demand queries over linked data. In *WWW*, 2010.
- [18] Andreas Harth, Jürgen Umbrich, Aidan Hogan, and Stefan Decker. YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In *ISWC/ASWC*, pages 211–224, 2007.
- [19] Olaf Hartig. How caching improves efficiency and result completeness for querying linked data. In *the 4th Linked Data on the Web (LDOW) Workshop at the World Wide Web Conference (WWW)*, 2011.

- [20] Olaf Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *Extended Semantic Web Conference*, 2011.
- [21] Olaf Hartig, Christian Bizer, and Johann Christoph Freytag. Executing sparql queries over the web of linked data. In *International Semantic Web Conference*, pages 293–309, 2009.
- [22] Stratos Idreos, Martin L. Kersten, and Stefan Manegold. Self-organizing tuple reconstruction in column-stores. In *ACM SIGMOD International Conference on Management of Data*, pages 297–308, 2009.
- [23] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.

- [24] Daniel Izquierdo, Maria-Esther Vidal, and Blai Bonet. An expressive and efficient solution to the service selection problem. In *International Semantic Web Conference (1)*, pages 386–401, 2010.
- [25] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *SIGMOD Conference*, pages 106–117, 1998.
- [26] Kamlesh Laddhad and S. Sudarshan. Adaptive query processing. Technical Report 05329014, Kanwal Rekhi School of Information Technology, Indian Institute of Technology, Bombay, Mumbai, 2006.
- [27] Gunter Ladwig and Thanh Tran. Linked data query processing strategies. In *International Semantic Web Conference (ISWC)*, 2010.

- [28] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. Robust query processing through progressive optimization. In *SIGMOD Conference*, pages 659–670, 2004.
- [29] Thomas Neumann and Gerhard Weikum. Scalable join processing on very large rdf graphs. In *SIGMOD International Conference on Management of Data*, pages 627–640, 2009.
- [30] Kuldeep B R Reddy and P Sreenivasa Kumar. Optimizing SPARQL Queries Over the Web of Linked Data. In *Proceedings of the International Workshop on Semantic Data Management (SemData 2010)*, 2010.
- [31] Edna Ruckhaus and Maria-Esther Vidal. The BAY-HIST Prediction Model for RDF Documents. In *Proceedings of the 2nd ESWC Workshop on Inductive Reasoning and Machine Learning on the Semantic Web-CEUR volume 611*, pages 30–41, 2010.

- [32] P. Selinger, M. Astrahan, D. Chamberlin, R. Lorie, and T. Price. Access Path Selection in a Relational Database Management System. *Proceedings of ACM Sigmod*, 1979.
- [33] Michael Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Samuel Madden, Elizabeth J. O'Neil, Patrick E. O'Neil, Alex Rasin, Nga Tran, and Stanley B. Zdonik. C-store: A column-oriented dbms. In *VLDB*, pages 553–564, 2005.
- [34] Michael Stonebraker, Paul M. Aoki, Witold Litwin, Avi Pfeffer, Adam Sah, Jeff Sidell, Carl Staelin, and Andrew Yu. Mariposa: A wide-area distributed database system. *VLDB J.*, 5(1):48–63, 1996.
- [35] Feng Tian and David J. DeWitt. Tuple routing strategies for distributed eddies. In *VLDB*, pages 333–344, 2003.

- [36] Tolga Urhan and Michael J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, 23(2):27–33, 2000.
- [37] Tolga Urhan and Michael J. Franklin. Dynamic pipeline scheduling for improving interactive query performance. In *VLDB*, pages 501–510, 2001.
- [38] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD Conference*, pages 130–141, 1998.
- [39] Maria-Esther Vidal, Edna Ruckhaus, Tomas Lampo, Amadis Martinez, Javier Sierra, and Axel Polleres. Efficiently Joining Group Patterns in SPARQL Queries. In *Proceedings of the 7th Extended Semantic Web Conference (ESWC2010)*, 2010.
- [40] Cathrin Weiss, Panagiotis Karras, and Abraham Bernstein. Hexastore: sextuple indexing for semantic web data management. *PVLDB*, 1(1):1008–1019, 2008.

- [41] Gio Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, 1992.
- [42] Eugene Wong and Karel Youssefi. Decomposition-a strategy for query processing. *ACM Trans. on Database Systems*, 1(3), 1976.
- [43] Vladimir Zadorozhny, Louiqa Raschid, Maria-Esther Vidal, Tolga Urhan, and Laura Bright. Efficient evaluation of queries in a mediator for websources. In *SIGMOD Conference*, pages 85–96, 2002.