Adaptive Semantic Data Management Techniques for Federations of Endpoints- Half-Day Tutorial at ESWC 2012

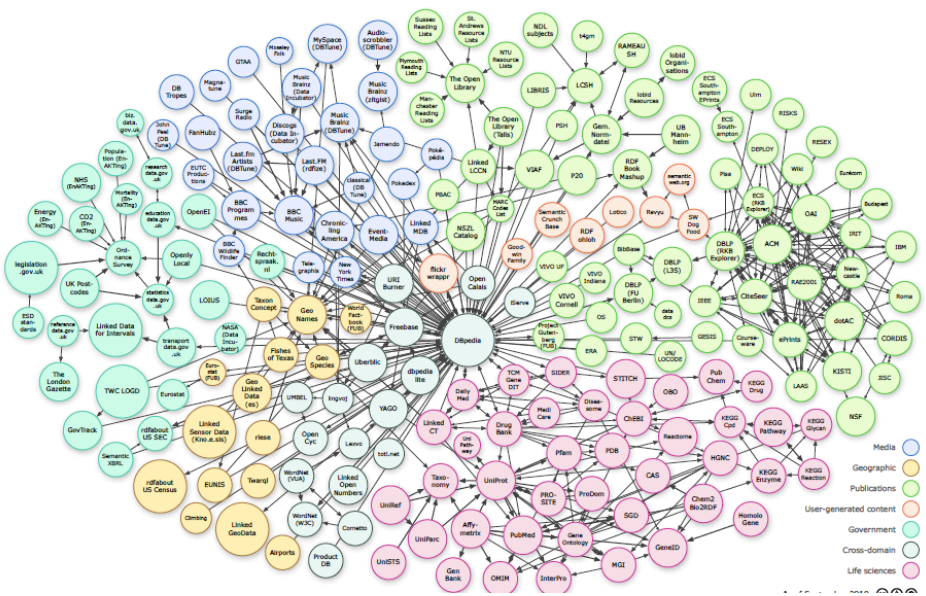
Maribel Acosta[1,2], Cosmin Basca[3], Gabriela Montoya[1]
Edna Ruckhaus[1],Maria-Esther Vidal[1]
[1] Universidad Simón Bolívar, Venezuela
{macosta,gmontoya,ruckhaus,mvidal}@ldc.usb.ve
[2] Institute AIFB, Karlsruhe Institute of Technology, Germany
Maribel.Acosta@aifb.uni-karlsruhe.de
[3] Department of Informatics, University of Zurich, Switzerland
basca@ifi.uzh.ch
http://www.ldc.usb.ve/~mvidal/TUTORIAL2012

May 28th, 2012

Introduction

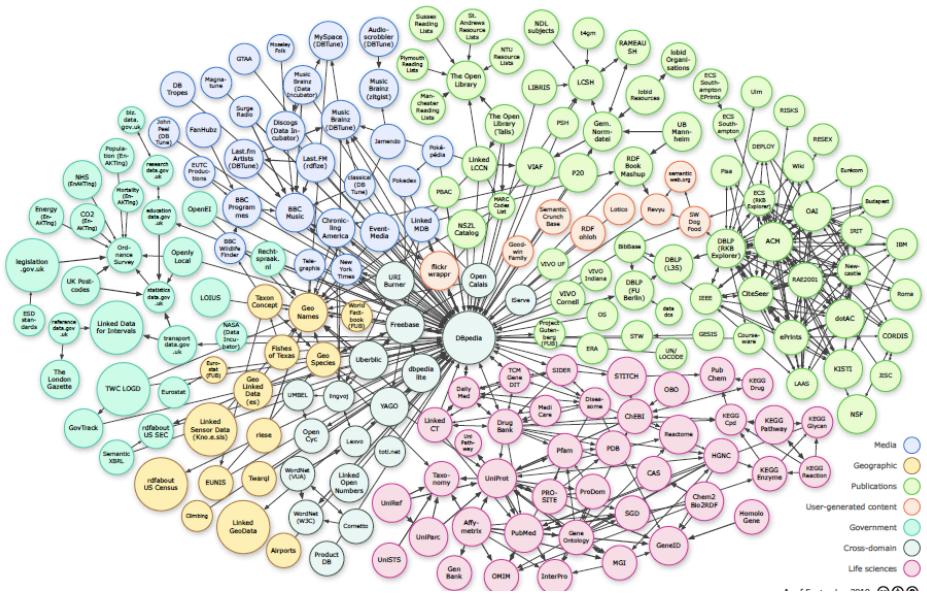
Motivation

- Cloud of Linked Data: a large number of datasets.
- Data locally or remotely accessed by RDF engines and SPARQL endpoints.
- RDF engines rely performance on physical access and storage structures that are locally stored, but:
  - loading the data and their links is not always feasible.
- The optimize-then-execute paradigm is usually followed, but:
  - lack of statistics.
  - unpredictable conditions of remote queries.
  - changing characteristics.



Motivation

- Cloud of Linked Data: a large number of datasets.
- Data locally or remotely accessed by RDF engines and SPARQL endpoints.
- RDF engines rely performance on physical access and storage structures that are locally stored, but:
  - loading the data and their links is not always feasible.
- The optimize-then-execute paradigm is usually followed, but:
  - lack of statistics.
  - unpredictable conditions of remote queries.
  - changing characteristics.



Techniques are not adaptable to unpredictable data transfers or data availability.

Motivating Example

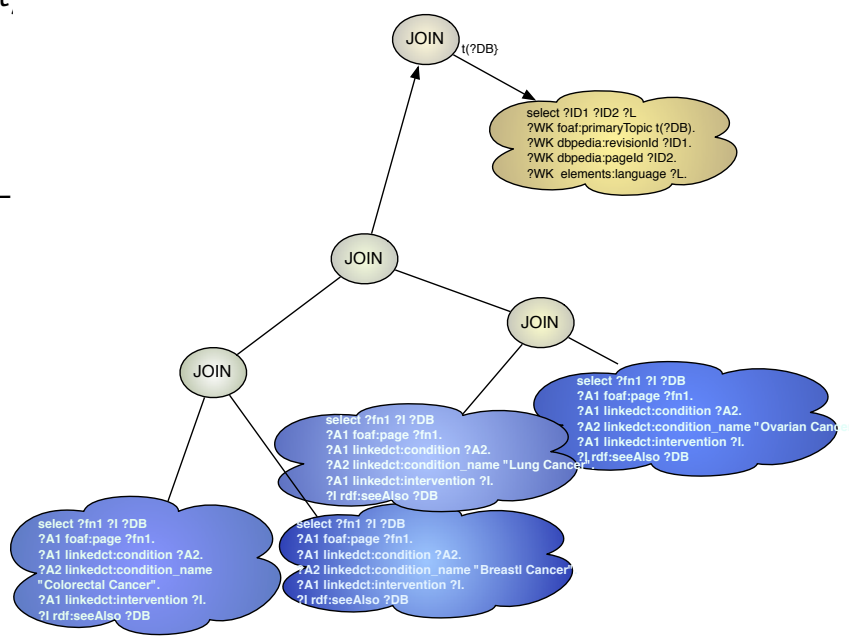
Clinical trials and drugs used for Breast, Colorectal, Ovarian, and Lung Cancer.

```
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.}
```

Motivating Example

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.

- Endpoints may time out before producing any answer.
- Query too complex, and needs to be decomposed into simple sub-queries.
- Adaptive query processing techniques need to be implemented to incrementally produce answers.

Agenda

1. Basic Concepts and Background (20 minutes)
2. Adaptive Query Processing Techniques (30 minutes)
3. RDF Engines for Federations of Endpoints (30 minutes)
4. Questions and Discussion (5 minutes)
5. Coffee Break (30 minutes)
6. Hands-on (90 minutes)
   - Explanation of existing Benchmarks (10 minutes)
   - Evaluation (60 minutes)
   - Discussion of the Observed Results (15 minutes)
7. Summary and Closing (5 minutes)

Background

Introduction-Traditional Data Management Systems[11, 16]

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

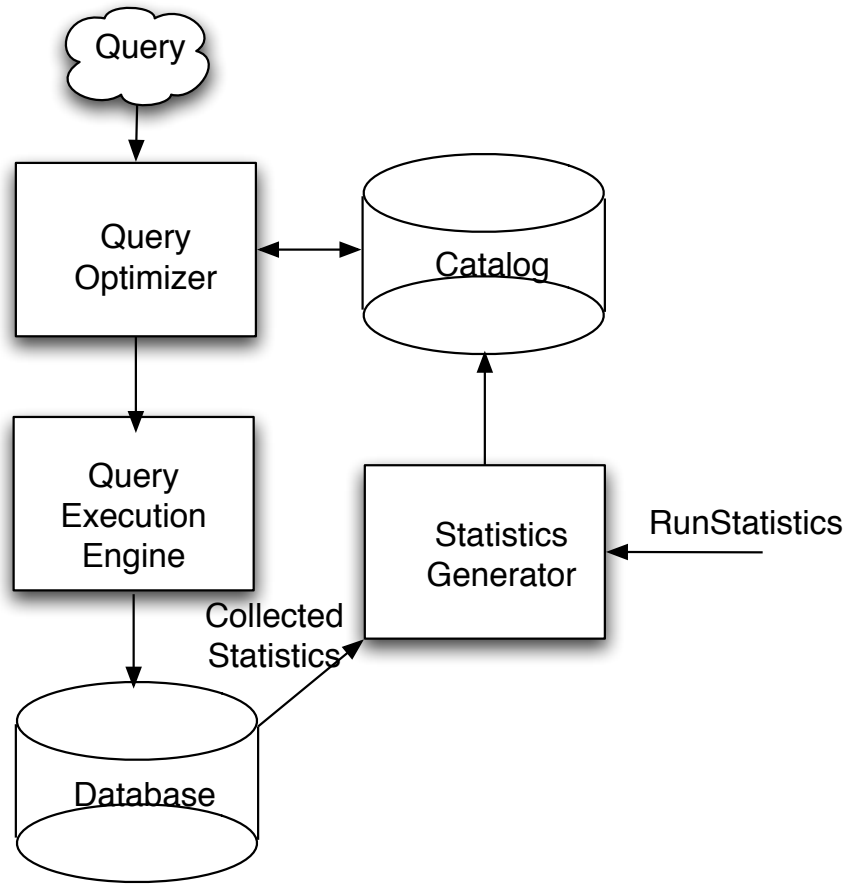
- consistently store and organize data,
  - Provide physical data structures to store large databases.
  - Stage large datasets between main memory and secondary storage.
  - Protect data from inconsistencies.
- and to provide a secure and concurrent access to the data.
  - Provide a query language.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

The Optimize-Then-Execute Paradigm

Traditional Architectures

Query Processing is divided into three mayor steps:

- Statistics generation.
- Query optimization.
- Query Execution.

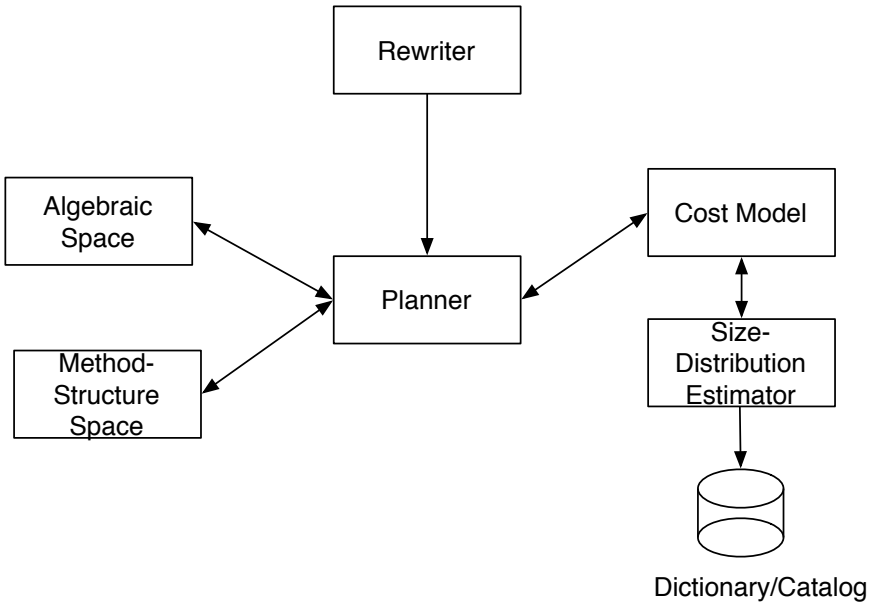


The Optimize-then-Execute Paradigm

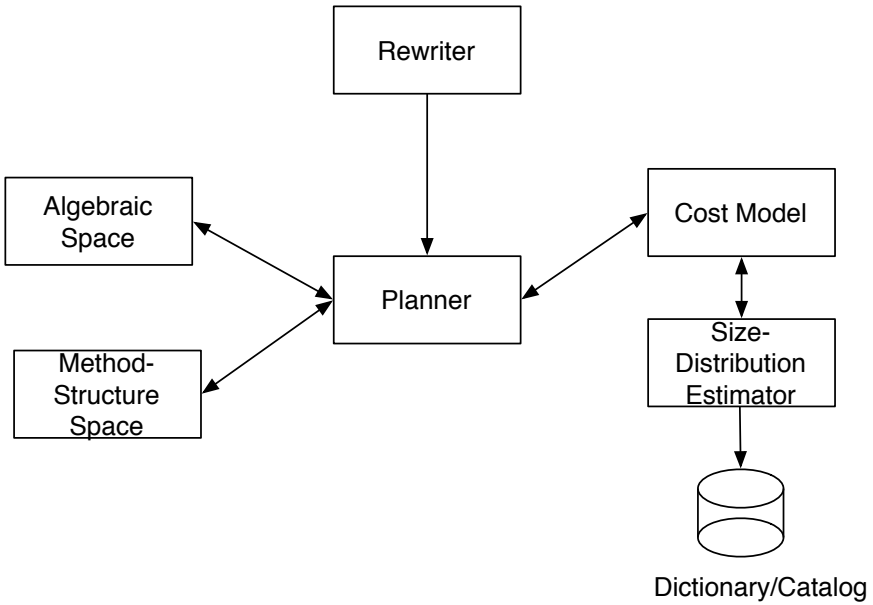
Traditional Query Processing techniques:

- Parse a declarative query.
- Generate an intermediate representation of the query.
- Produce an efficient logical and physical plan; minimize disk I/O access.
- Execute the query plan without making runtime decisions.

Query Optimizer Architecture[11, 16, 22]



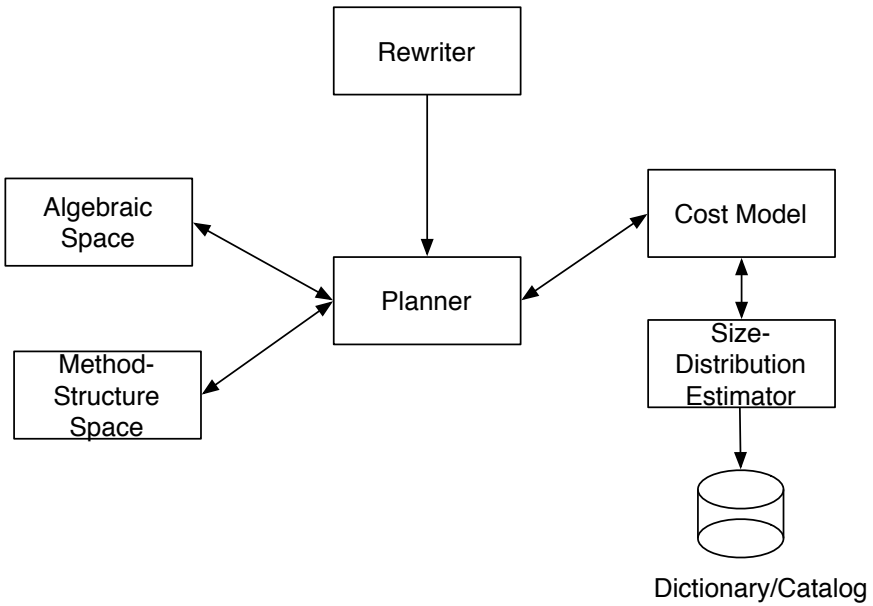
Query Optimizer Architecture[11, 16, 22]



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*.

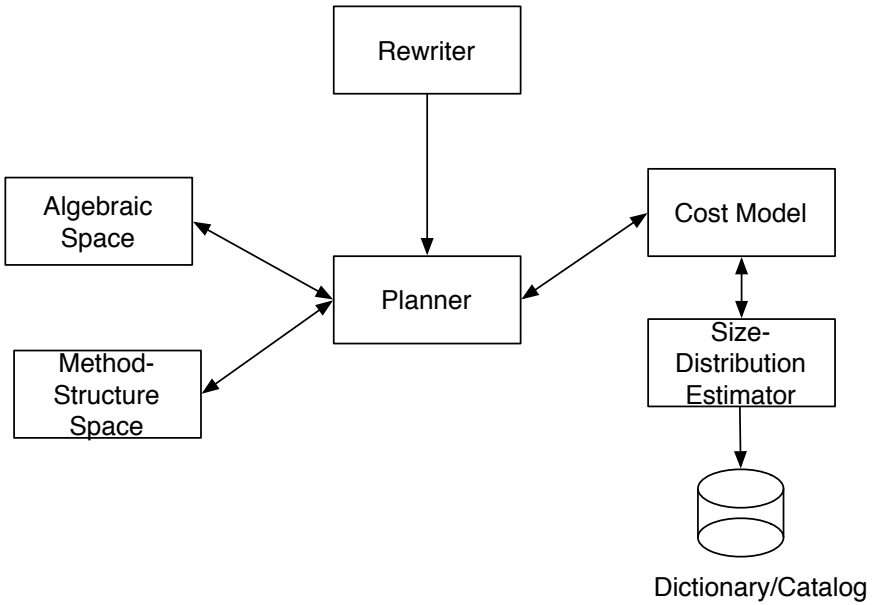
Query Optimizer Architecture[11, 16, 22]



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.

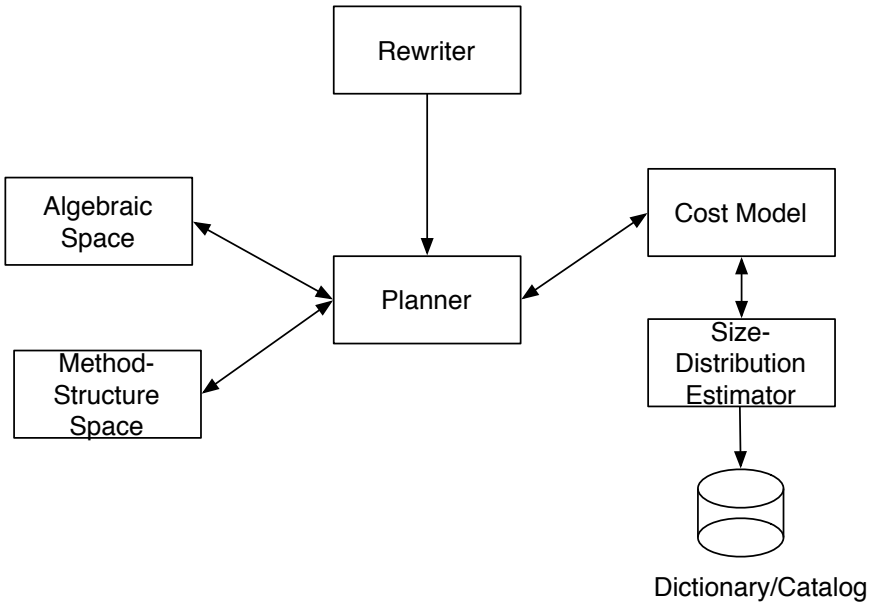
Query Optimizer Architecture[11, 16, 22]



Algebraic Space

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

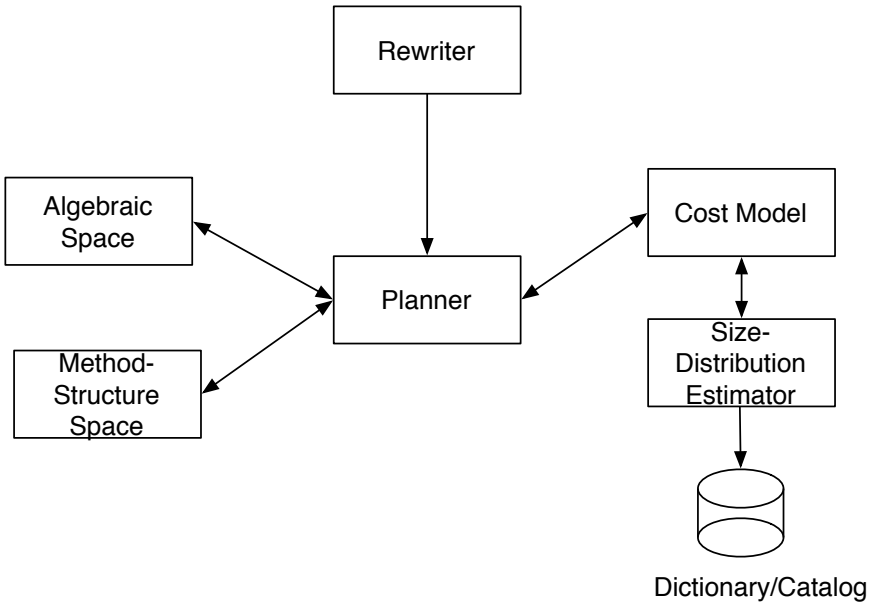
Query Optimizer Architecture[11, 16, 22]



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.

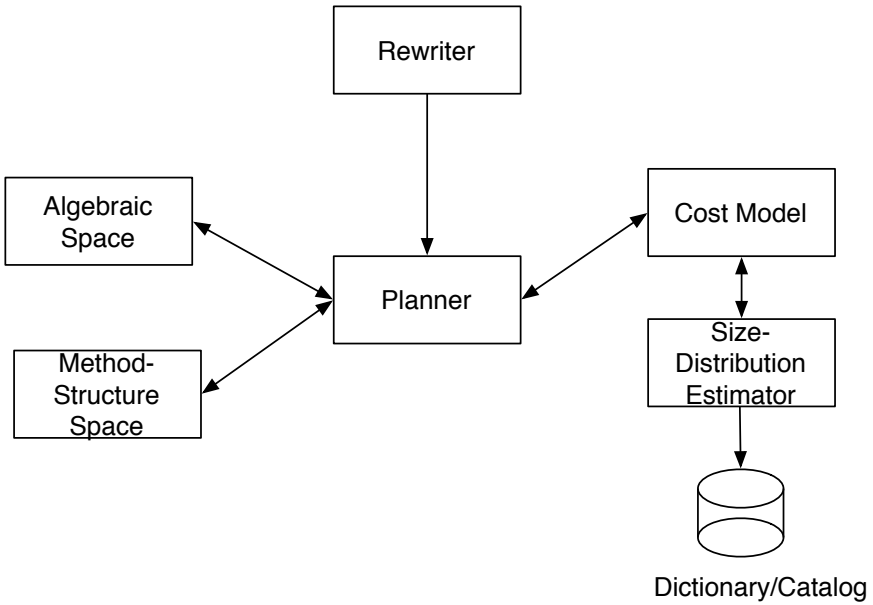
Query Optimizer Architecture[11, 16, 22]



Cost Model

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

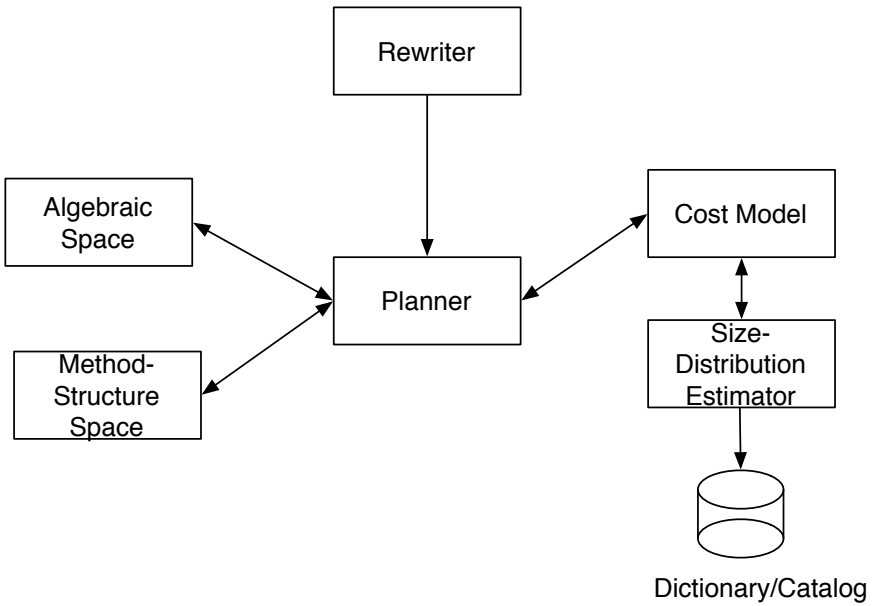
Query Optimizer Architecture[11, 16, 22]



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.

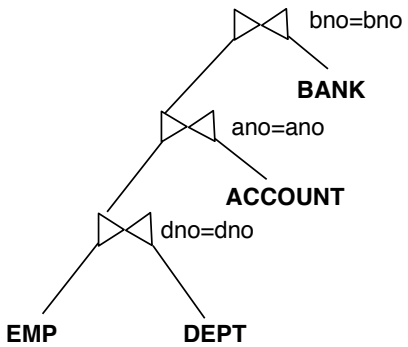
Query Optimizer Architecture[11, 16, 22]



Dictionary

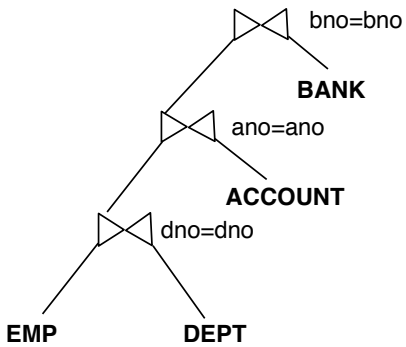
Set of statistics that characterize the dataset, e.g., number of different values of an attribute, or different subjects, properties, or values; distributions, etc. Stored statistics depend on the type of size-distribution estimator.

Join Ordering: Search Space

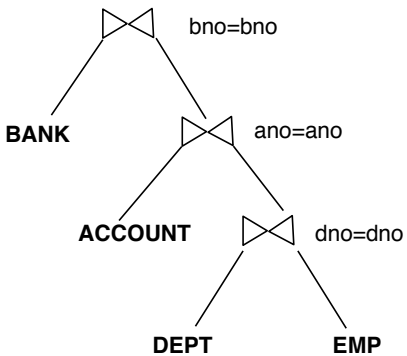


Left-linear plan

Join Ordering: Search Space

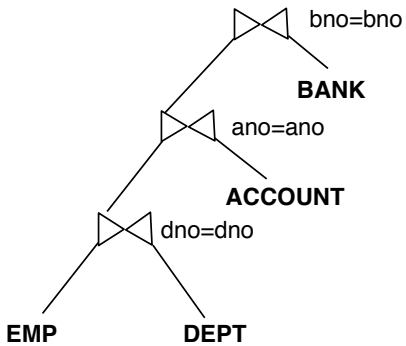


Left-linear plan

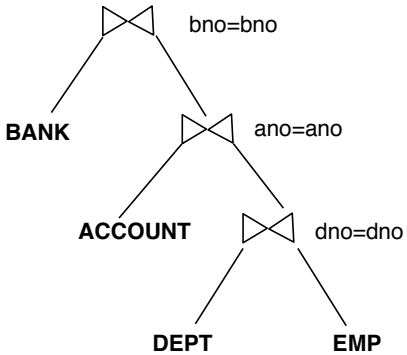


Right-linear plan

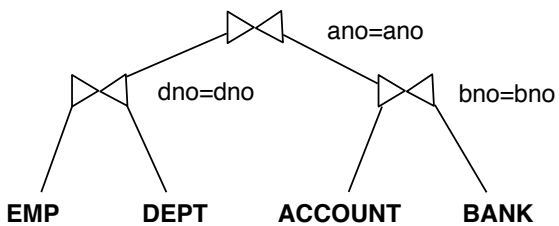
Join Ordering: Search Space



Left-linear plan



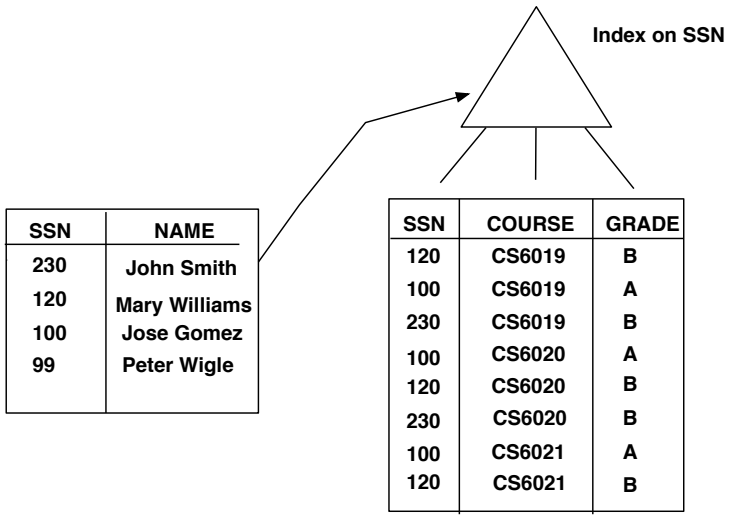
Right-linear plan



Bushy Plan

Index Nested Loop Join

- 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.



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.

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

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.

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

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

Sorting 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.

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

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

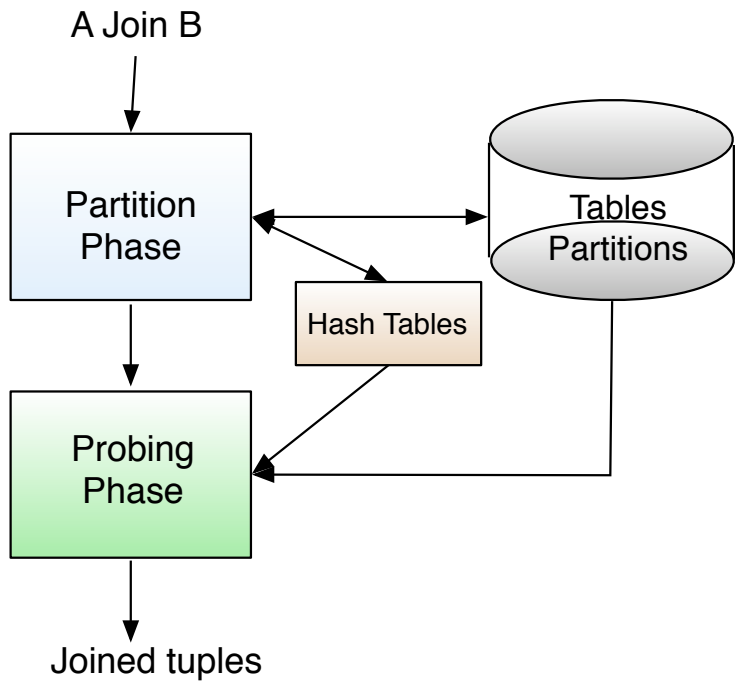
Sorting by SSN

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

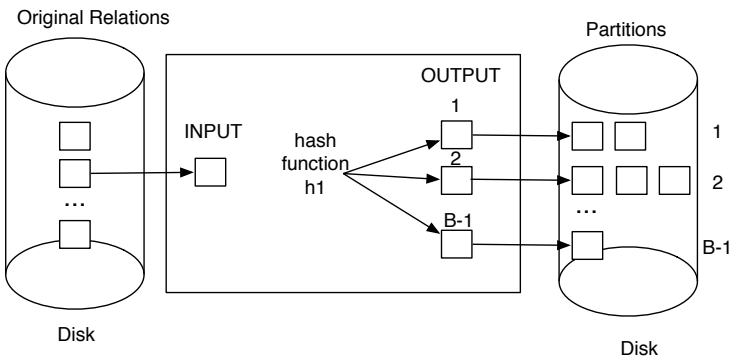
Merging by SSN

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.

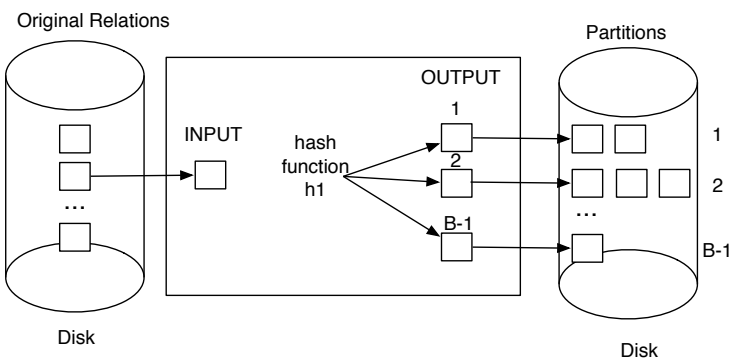


Example Hash Join

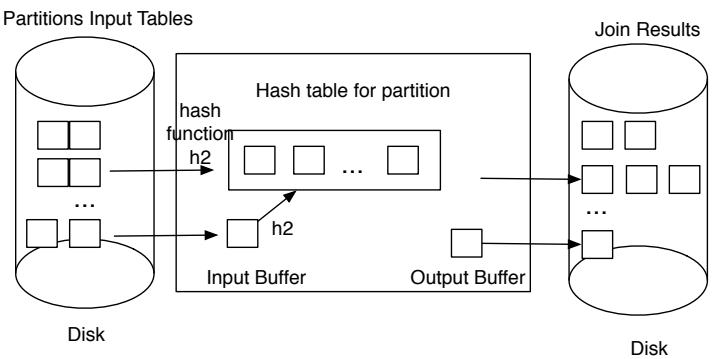


Partition Phase

Example Hash Join



Partition Phase

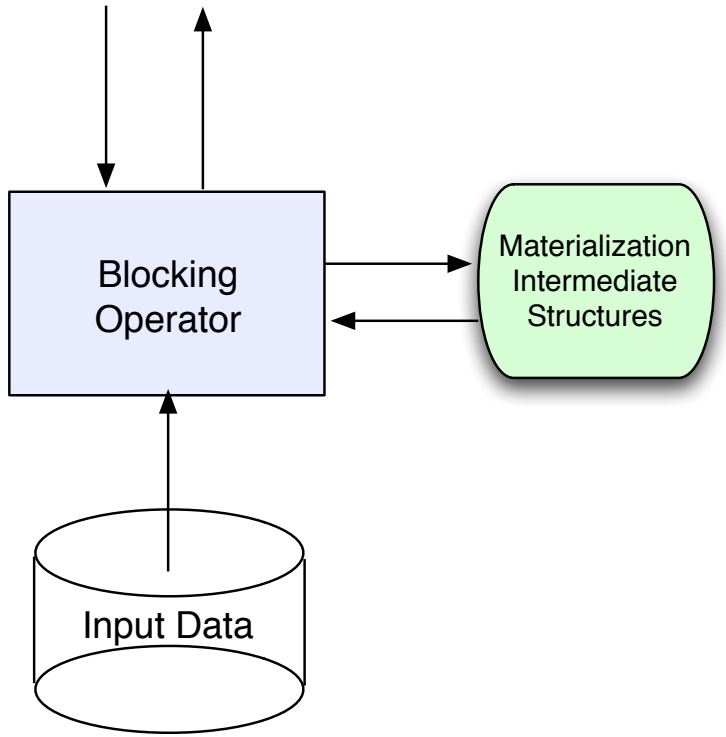


Probing Phase

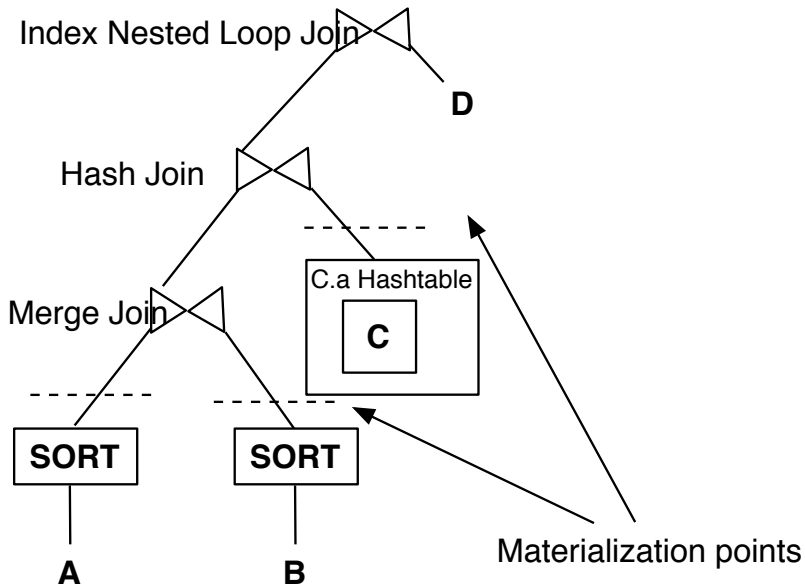
Execution Engine-Blocking Operators

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

  Project: must sort and then merge the input.
  Sort Merge Join: must sort the inputs and then merge them.
  Hash Join: must partition the inputs, 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 [11]

Iterator	*Open*	*Next*	*Close*
Scan	*open* input	call *next* on input	*close* input

Examples of Iterator Functions [11]

Iterator	*Open*	*Next*	*Close*
Scan	*open* input	call *next* on input	*close* input
Select	*open* input	call *next* on input until an item qualifies	*close* input

Examples of Iterator Functions [11]

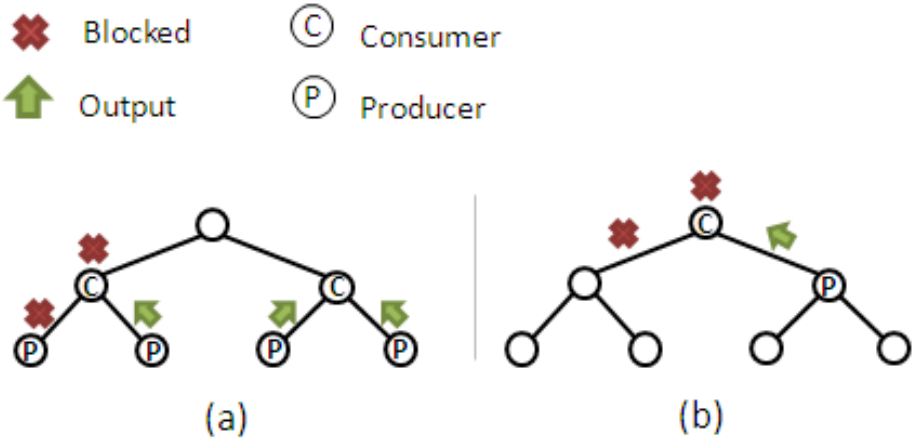
Iterator	*Open*	*Next*	*Close*
Scan	*open* input	call *next* on input	*close* input
Select	*open* input	call *next* on input until an item qualifies	*close* input
Hash Join	allocate hash directory open left *partition* input build hash table *next* on *partition* input *close partition* input *open*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

Examples of Iterator Functions [11]

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

Limitations of the Optimize-Then-Execute Paradigm Three Mayor Steps

- Off-line statistics generation to collect cardinalities, values frequencies, and costs to access data.
- Cost and heuristic-based query optimization to identify "good-enough" plans.
  Assumptions:
  - Different values of the attributes
  - Availability of the data
  - Cardinality of operators
  - Behavior of data sources
- Query Execution:
  - pipelined operators are processed at the time and output is produced by its sub-operators.
  - materialized operators are completely executed before firing its parents for processing.



Blocked Iterator Model.

Summary: The Optimize-Then-Execute Paradigm

- Relies on knowledge about the cost and cardinality of the accessed datasets.
- Effectively achieves the generation of query execution plans where the size of intermediate results is minimized.
- Does not scale due to lack of statistics and unpredictable data transfers or data availability.

The Adaptive Query Processing Paradigm

Adaptive Query Processing [4, 9, 18]

Adaptivity is needed:

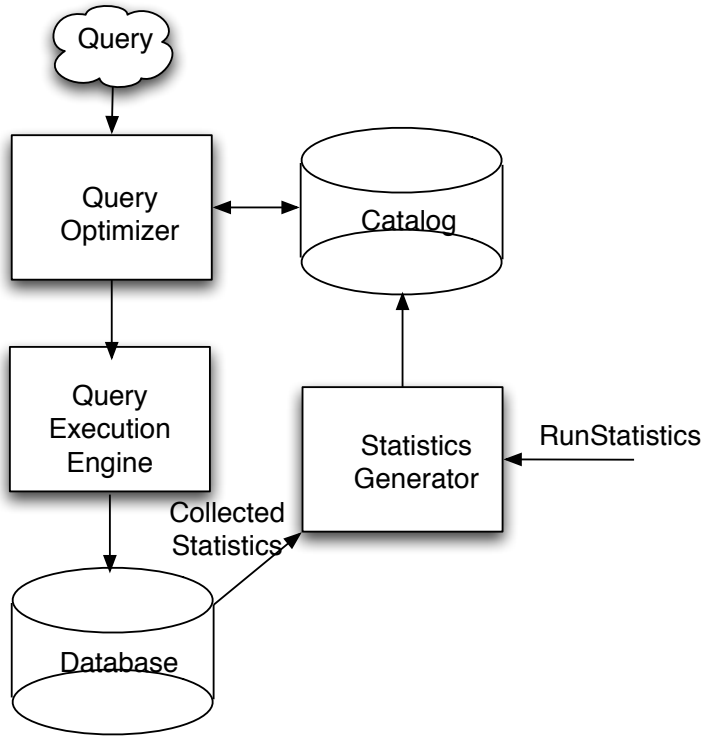
- Misestimated or missing statistics.
- Unexpected correlations.
- Unpredictable costs.
- Dynamically changing data, workload and sources availability.
- Rates at which tuples arrive from the sources, change.

  Initial Delays.
  Slow Delivery
  Bursty Arrivals
- Iterative environments.

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

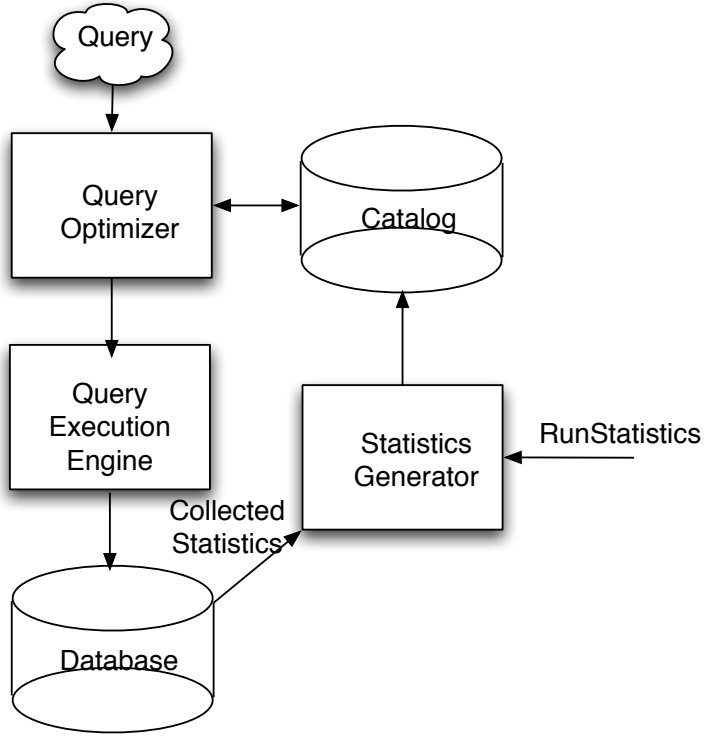
- Receive information from the environment.
- Use up-to-date information to change their behavior.
- Keep iterating over time to adapt their behavior based on the environment conditions.

Adaptive Query Processing

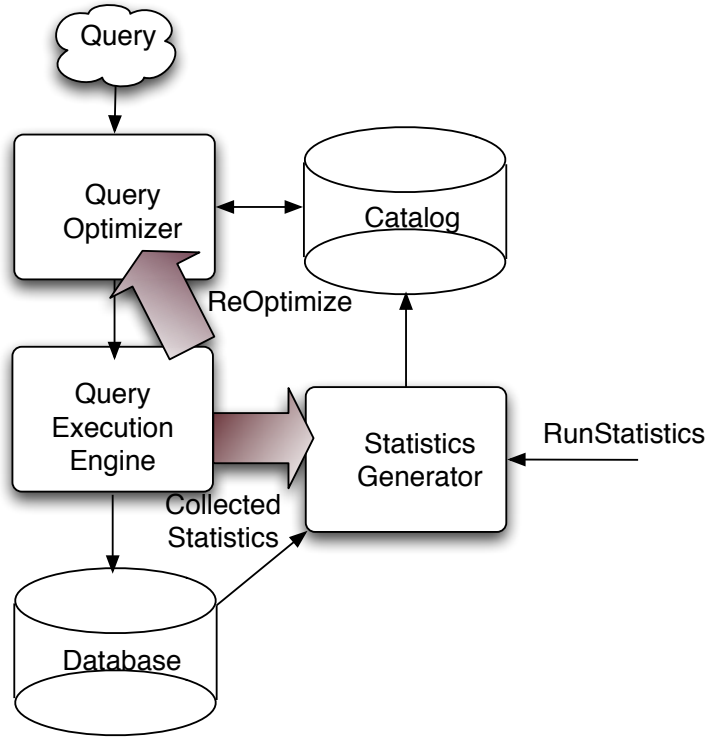


Typical Optimize-Then-Execute Architecture.

Adaptive Query Processing



Typical Optimize-Then-Execute Architecture.



Adaptive Query Processing Architecture.

Agenda

- Types of Adaptation.
- Adaptivity in Relational Databases
- Adaptivity in the Web of Data

Types of Adaptive-Based Solution

- Adaptation Level
  - Source Selection searching strategies to select the best sources for answering a query according to real-time source conditions.
  - Query Execution strategies to adapt query processing to environment conditions.
- Granularity of the Adaptation
  - Fine-grained adaptation of small processes, e.g., per-tuple or source basis.
  - Coarse-grained is attempted for large processes, e.g., several queries are executed under certain assumptions about the conditions of the environment.

Adaptive Query Processing Strategies

Plan-based Systems adapt to environment conditions, by extending the optimize-then-execute paradigm.

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 Re-optimization) 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.

Inter-Operator Re-optimization[4, 17, 21, 27]

- Ingress [30] interleaves sub-plan generation with plan execution; for each table in a query, the smallest tables are selected first until plans of all the tuples and tables are considered.
- Dynamic Query Planning [7] off-line define a set of constraints that optimal plans must meet; during execution time, plans that meet constraints are generated based on dataset conditions.
- DEC RDB [2] implements *competition*, an adaptive strategy which generates and runs multiple plans in parallel for a while, and after a short period of time it selects the most promising ones and discards the rest.
- Query Scrambling [27, 28] starts with a plan and it may change it on-the-fly, depending on data transfer delays or misestimations; operators can be ordered differently, or not directly combined tables can be joined by a new operator during re-optimization.
- Routing techniques avoid traditional optimizers, and generate plans for each tuple during runtime by sending tuples through the pool of eligible operators; Eddy [3, 25], M-Join [29] and STAIRs [8] follow this adaptive strategy.

Fine-Grained adaptivity is achieved.

Inter-Operator Re-optimization[4, 17, 21, 27]

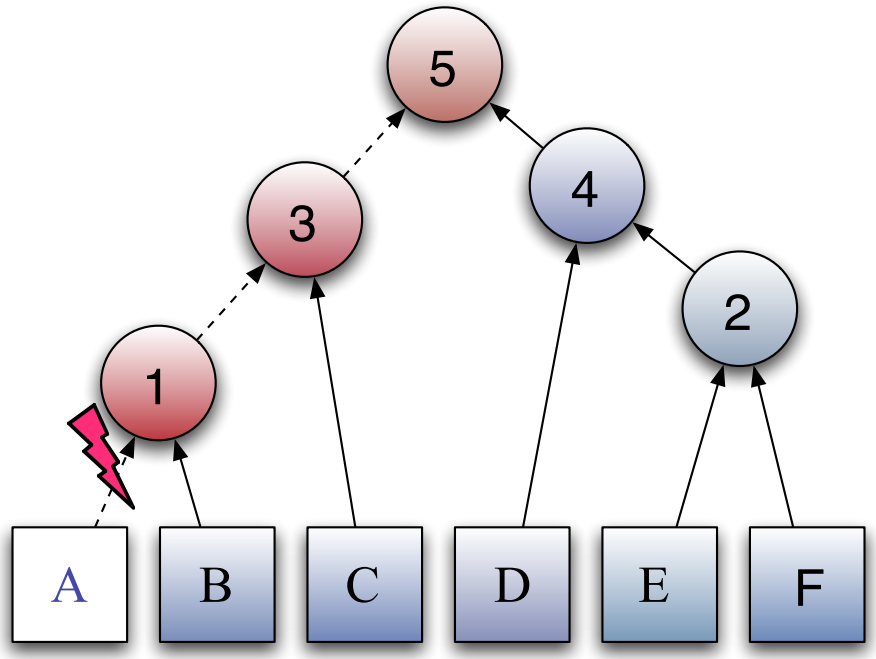
- Detect and correct situations where optimizers select sub-optimal plans, which could be improved if accurate statistics are considered.
- Existing approaches extend statistics tracking with a log of estimates considered by the optimizer to generate the query plan, and of system conditions during query 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[27, 28]

- Query Scrambling is an execution query method that reacts to data arrival delays.
- First, a plan is generated, and reacts during query execution to misestimations.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable sub-trees are identified, i.e., sub-tree 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[27, 28]

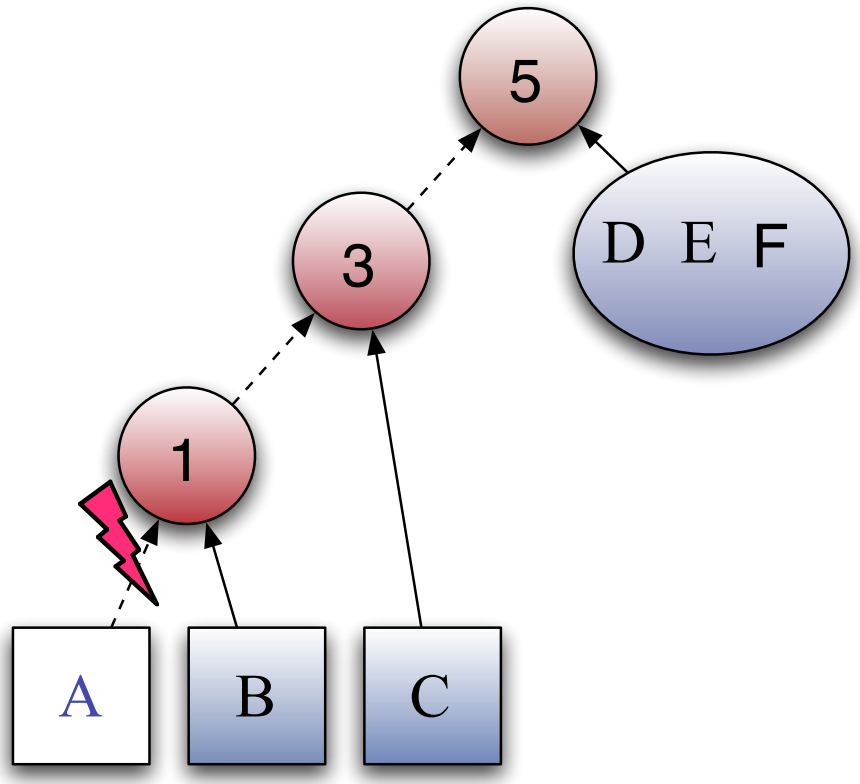
- Query Scrambling is an execution query method that reacts to data arrival delays.
- First, a plan is generated, and reacts during query execution to misestimations.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable sub-trees are identified, i.e., sub-tree 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.



Source A is delayed.

Query Scrambling[27, 28]

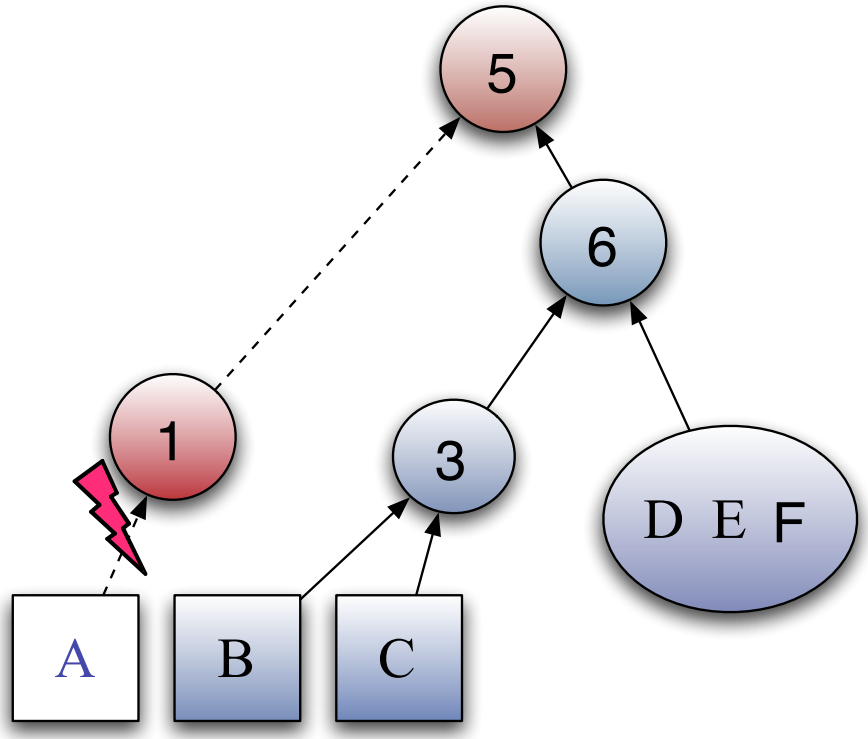
- Query Scrambling is an execution query method that reacts to data arrival delays.
- First, a plan is generated, and reacts during query execution to misestimations.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable sub-trees are identified, i.e., sub-tree 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.



Sub-tree 4 is materialized.

Query Scrambling[27, 28]

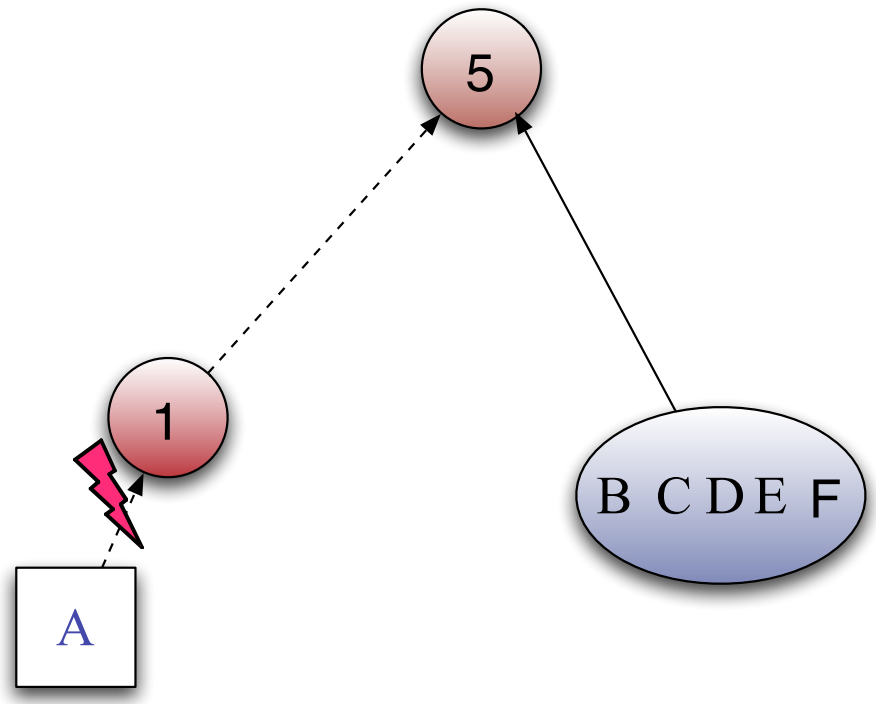
- Query Scrambling is an execution query method that reacts to data arrival delays.
- First, a plan is generated, and reacts during query execution to misestimations.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable sub-trees are identified, i.e., sub-tree 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.



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

Query Scrambling[27, 28]

- Query Scrambling is an execution query method that reacts to data arrival delays.
- First, a plan is generated, and reacts during query execution to misestimations.
- Existing operators can be rescheduled, retaining one part of the plan, and scheduling another.
- Query operators are divided into runnable and blocked.
- Maximal runnable sub-trees are identified, i.e., sub-tree 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.



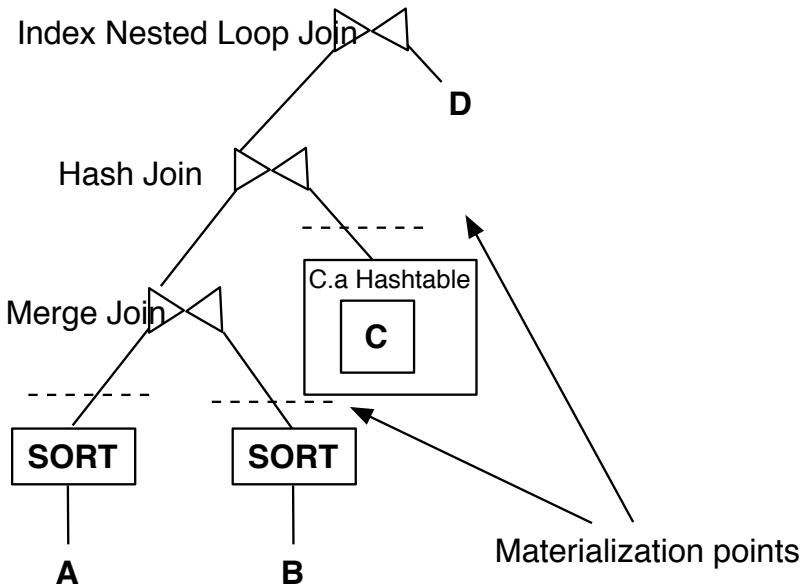
The new operator is executed.

Intra-Operator Techniques

- Adapts plans to environment changes, even in presence of a fixed plan.
- Incrementally produce results as they become available.
- Symmetric Hash Join [9] and Xjoin [26] overcome Hash Join blocking limitations, and hide delays while producing answers even if both input data sources become blocked.

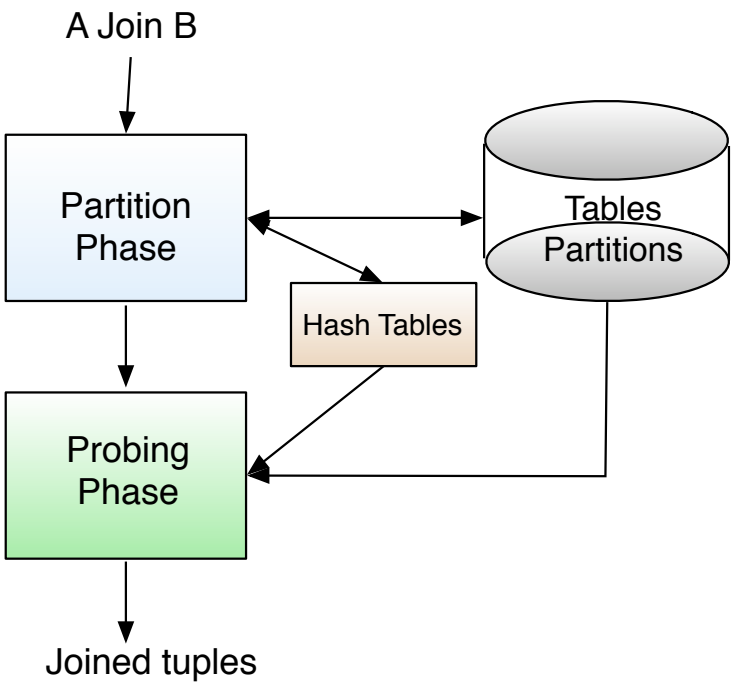
Fine-Grained adaptivity is achieved at each operator.

Example-Blocking Operators



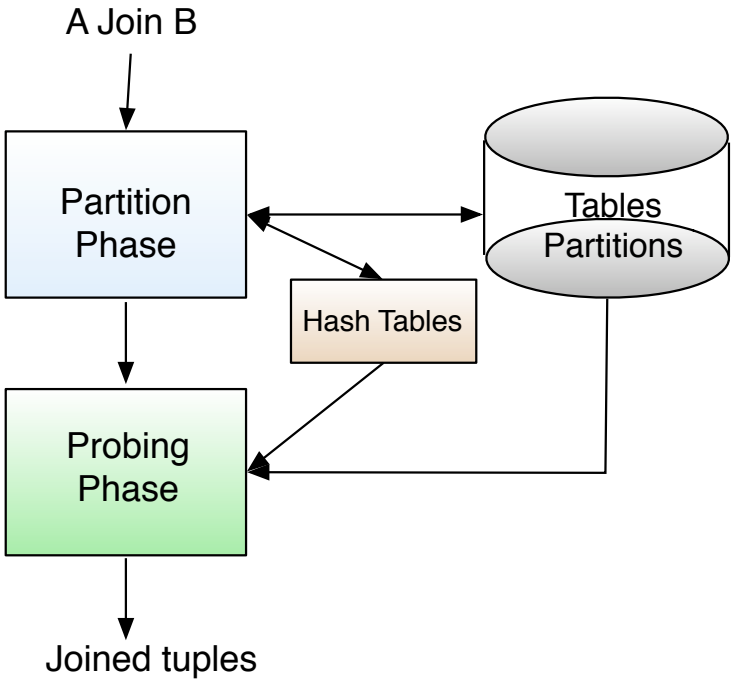
Blocking Operators

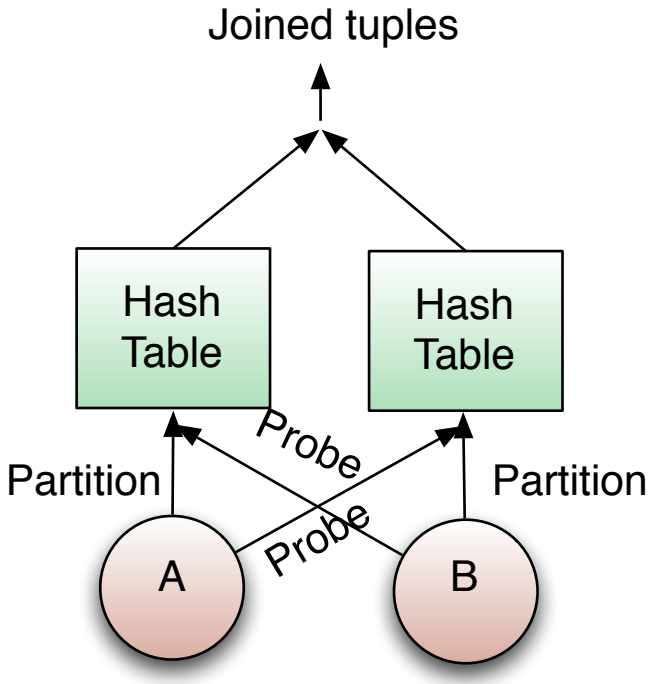
Symmetric Hash Join[9]



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

Symmetric Hash Join[9]





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.

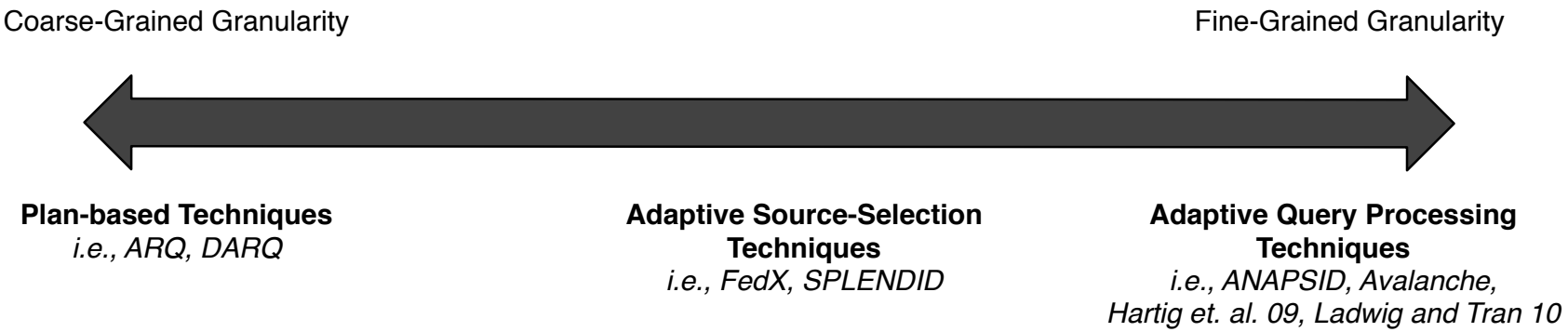
Adaptivity in the Web of Data

Adaptive Query Processing for the Web of Data

Adaptivity during Source Selection relies on source or endpoint descriptions and statistics describe data stored at the endpoints.

Adaptivity at Query Execution Level approaches implement intra- and inter-operator query processing techniques to adapt execution to no accurate statistics, uncontrollable network conditions, query complexity, dynamic data and workload.

Evolution of Adaptive Approaches in the Web of Data Summary



Adaptivity during Source Selection coarse-Grained Approaches

- Harth et al.[12]: Qtree-based index structure to store data source statistics collected in a pre-processing stage; these statistics are used for source ranking and to determine the best sources to answer a join query.
- Li and Heflin[20] tree structure to support integration of data from multiple heterogeneous sources; nodes are annotated with source join selectivities and used to select the corresponding datasets.

Finer granularity is achieved whenever the statistics are re-computed frequently.

Fined-Grained Approaches to traverse the Web of Data

- Hartig et al.[13, 14, 15] source selection and link traversal are interleaved during query execution time. A non-blocking iterator model is used for traversing relevant links
  - relies on an asynchronous pipeline of iterators executed in an order heuristically determined. e.g., the most selective iterators are executed first [14].
  - query engine is able to adapt the execution to source availability by detecting on-the-fly whenever a dereferenced dataset stops responding; iterators can be cancelled, and other requests are submitted to alive datasets.

Fine-Grained adaptivity is achieved during query selection and link traversal.

Combined-Grained Approaches to traverse the Web of Data

- Ladwig and Tran[19]

  coarse-grained granularity: aggregate indexes keep data distributions and are used for source selection.

  Fine-grained granularity: Symmetric Hash Joins incrementally produce answers even when sources become blocked.

Adaptivity is managed at different levels and granularity during link traversal, but these approaches are not tailored to access data from federations of endpoints.

SPARQL Endpoints

- Implement *SPARQL protocol* and enable users to *query particular datasets* or to dereference linked data.
- Developed for very *light-weight* use.
- *Executions* against several endpoints can be *unsuccessful* because of endpoint timeouts, unpredictable data transfers and endpoint availability.

SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions[10]

- Statistical Information stored in VOID Descriptions is used to perform source selection first stage.
- Contact Source during query execution is used to improve the preliminar source selection.
- Pattern grouping is performed for set of triples that share the same unique endpoint, and for *sameAs* triple that shares an unbound subject variable with other triples.
- Cost-based join ordering using dynamic programming determines the best plan, giving preference to bushy tree plans.
- Bind Joins that reduces network overhead.

SPLENDID

Example: drugs and their components' url and image

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX bio2rdf:<http://bio2rdf.org/ns/bio2rdf#>
SELECT $drug $keggUrl $chebiImage WHERE {
      $drug rdf:type drugbank:drugs .
      $drug drugbank:keggCompoundId $keggDrug .
      $drug drugbank:genericName $drugBankName .
      $keggDrug bio2rdf:url $keggUrl .
      $chebiDrug purl:title $drugBankName .
      $chebiDrug bio2rdf:image $chebiImage .

}
```

SPLENDID

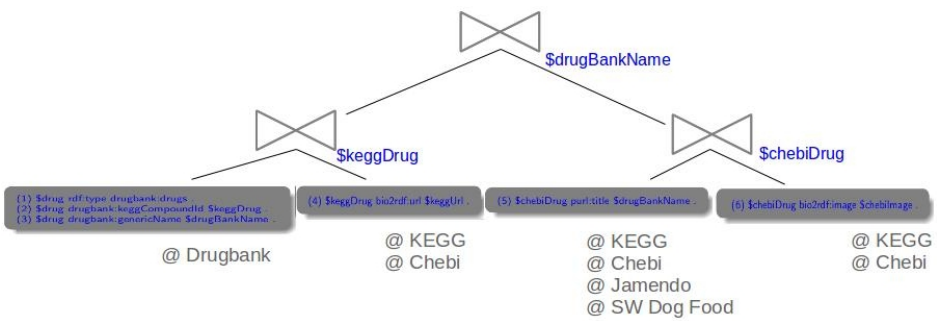
Example: drugs and their components' url and image

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX bio2rdf:<http://bio2rdf.org/ns/bio2rdf#>
SELECT $drug $keggUrl $chebiImage WHERE {
  (1) $drug rdf:type drugbank:drugs .
  (2) $drug drugbank:keggCompoundId $keggDrug .                    Drugbank
  (3) $drug drugbank:genericName $drugBankName .
  (4) $keggDrug bio2rdf:url $keggUrl .                             KEGG, Chebi
  (5) $chebiDrug purl:title $drugBankName .                        KEGG, Chebi, Jamendo, SW Dog Food
  (6) $chebiDrug bio2rdf:image $chebiImage .                       KEGG, Chebi
}
```

SPLENDID

Example: drugs and their components' url and image

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX bio2rdf:<http://bio2rdf.org/ns/bio2rdf#>
SELECT $drug $keggUrl $chebiImage WHERE {
  (1) $drug rdf:type drugbank:drugs .
  (2) $drug drugbank:keggCompoundId $keggDrug .                    Drugbank
  (3) $drug drugbank:genericName $drugBankName .
  (4) $keggDrug bio2rdf:url $keggUrl .                             KEGG, Chebi
  (5) $chebiDrug purl:title $drugBankName .                        KEGG, Chebi, Jamendo, SW Dog Food
  (6) $chebiDrug bio2rdf:image $chebiImage .                       KEGG, Chebi
}
```



Execution plan

SPLENDID

Example: drugs that interact with antibiotics, antiviral and antihypertensive agents

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dbcategory: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbowl: <http://dbpedia.org/ontology/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
SELECT DISTINCT ?drug ?enzyme ?reaction Where {
    ?drug1 drugbank:drugCategory dbcategory:antibiotics .
    ?drug2 drugbank:drugCategory dbcategory:antiviralAgents .
    ?drug3 drugbank:drugCategory dbcategory:antihypertensiveAgents .
    ?I1 drugbank:interactionDrug2 ?drug1 .
    ?I1 drugbank:interactionDrug1 ?drug .
    ?I2 drugbank:interactionDrug2 ?drug2 .
    ?I2 drugbank:interactionDrug1 ?drug .
    ?I3 drugbank:interactionDrug2 ?drug3 .
    ?I3 drugbank:interactionDrug1 ?drug .
    ?drug owl:sameAs ?drug5 .
    ?drug drugbank:keggCompoundId ?cpd .
    ?enzyme kegg:xSubstrate ?cpd .
    ?enzyme rdf:type kegg:Enzyme .
    ?reaction kegg:xEnzyme ?enzyme .
    ?reaction kegg:equation ?equation .
    ?drug5 rdf:type dbowl:Drug .

}
```

SPLENDID

Example: drugs that interact with antibiotics, antiviral and antihypertensive agents

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dbcategory: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbowl: <http://dbpedia.org/ontology/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
SELECT DISTINCT ?drug ?enzyme ?reaction Where {
```

```
(1)  ?drug1 drugbank:drugCategory dbcategory:antibiotics .
(2)  ?drug2 drugbank:drugCategory dbcategory:antiviralAgents .
(3)  ?drug3 drugbank:drugCategory dbcategory:antihypertensiveAgents .
(4)  ?I1 drugbank:interactionDrug2 ?drug1 .
(5)  ?I1 drugbank:interactionDrug1 ?drug .
(6)  ?I2 drugbank:interactionDrug2 ?drug2 .
(7)  ?I2 drugbank:interactionDrug1 ?drug .
(8)  ?I3 drugbank:interactionDrug2 ?drug3 .
(9)  ?I3 drugbank:interactionDrug1 ?drug .
(10) ?drug owl:sameAs ?drug5 .
(11) ?drug drugbank:keggCompoundId ?cpd .
```

Drugbank

```
(12) ?enzyme kegg:xSubstrate ?cpd .
(13) ?enzyme rdf:type kegg:Enzyme .
(14) ?reaction kegg:xEnzyme ?enzyme .
(15) ?reaction kegg:equation ?equation .
```

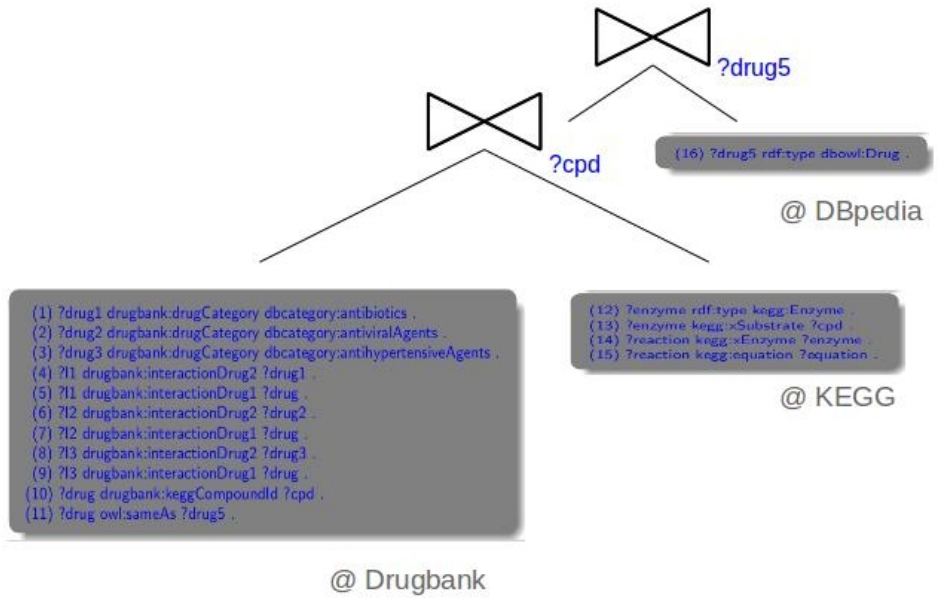
KEGG

```
(16) ?drug5 rdf:type dbowl:Drug .
```

DBpedia

```
}
```

SPLENDID



Execution plan

Adaptivity at Endpoint Selection Level Fine-Grained Granularity

FedX [24]

- no knowledge about mappings or statistics is maintained.
- endpoints are consulted on-the-fly to determine if a predicates can be answered.
- cache is used to record endpoint predicates.
- queries are divided into exclusive groups, i.e., triple patterns that can be executed by only one endpoint.
- heuristically groups are ordered.

Fine-grained granularity at source selection level while coarse-grained granularity during plan generation.

FedX

Example: drugs and their components' url and image

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX bio2rdf:<http://bio2rdf.org/ns/bio2rdf#>
SELECT $drug $keggUrl $chebiImage WHERE {
      $drug rdf:type drugbank:drugs .
      $drug drugbank:keggCompoundId $keggDrug .
      $drug drugbank:genericName $drugBankName .
      $keggDrug bio2rdf:url $keggUrl .
      $chebiDrug purl:title $drugBankName .
      $chebiDrug bio2rdf:image $chebiImage .

}
```

FedX

Example: drugs and their components' url and image

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX bio2rdf:<http://bio2rdf.org/ns/bio2rdf#>
SELECT $drug $keggUrl $chebiImage WHERE {
  (1) $drug rdf:type drugbank:drugs .
  (2) $drug drugbank:keggCompoundId $keggDrug .                    Drugbank
  (3) $drug drugbank:genericName $drugBankName .

  (4) $keggDrug bio2rdf:url $keggUrl .                             KEGG, Chebi

  (5) $chebiDrug purl:title $drugBankName .                        KEGG, Chebi, Jamendo, SW Dog Food

  (6) $chebiDrug bio2rdf:image $chebiImage .                       KEGG, Chebi

}
```

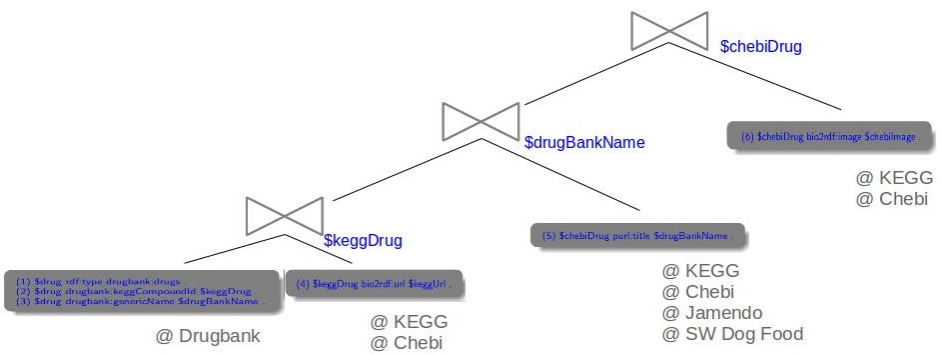
FedX

Example: drugs and their components' url and image

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX purl:<http://purl.org/dc/elements/1.1/>
PREFIX bio2rdf:<http://bio2rdf.org/ns/bio2rdf#>
SELECT $drug $keggUrl $chebiImage WHERE {
```

(1)	$drug rdf:type drugbank:drugs .	Drugbank
(2)	$drug drugbank:keggCompoundId $keggDrug .	
(3)	$drug drugbank:genericName $drugBankName .	
(4)	$keggDrug bio2rdf:url $keggUrl .	KEGG, Chebi
(5)	$chebiDrug purl:title $drugBankName .	KEGG, Chebi, Jamendo, SW Dog Food
(6)	$chebiDrug bio2rdf:image $chebiImage .	KEGG, Chebi

```
}
```



Execution plan

FedX

Example: drugs that interact with antibiotics, antiviral and antihypertensive agents

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dbcategory: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbowl: <http://dbpedia.org/ontology/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
SELECT DISTINCT ?drug ?enzyme ?reaction Where {
    ?drug1 drugbank:drugCategory dbcategory:antibiotics .
    ?drug2 drugbank:drugCategory dbcategory:antiviralAgents .
    ?drug3 drugbank:drugCategory dbcategory:antihypertensiveAgents .
    ?I1 drugbank:interactionDrug2 ?drug1 .
    ?I1 drugbank:interactionDrug1 ?drug .
    ?I2 drugbank:interactionDrug2 ?drug2 .
    ?I2 drugbank:interactionDrug1 ?drug .
    ?I3 drugbank:interactionDrug2 ?drug3 .
    ?I3 drugbank:interactionDrug1 ?drug .
    ?drug drugbank:keggCompoundId ?cpd .
    ?enzyme kegg:xSubstrate ?cpd .
    ?enzyme rdf:type kegg:Enzyme .
    ?reaction kegg:xEnzyme ?enzyme .
    ?reaction kegg:equation ?equation .
    ?drug5 rdf:type dbowl:Drug .
    ?drug owl:sameAs ?drug5 .

}
```

FedX

Example: drugs that interact with antibiotics, antiviral and antihypertensive agents

```
PREFIX drugbank: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugbank/>
PREFIX dbcategory: <http://www4.wiwiss.fu-berlin.de/drugbank/resource/drugcategory/>
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX dbowl: <http://dbpedia.org/ontology/>
PREFIX kegg: <http://bio2rdf.org/ns/kegg#>
SELECT DISTINCT ?drug ?enzyme ?reaction Where {
(1)  ?drug1 drugbank:drugCategory dbcategory:antibiotics .
(2)  ?drug2 drugbank:drugCategory dbcategory:antiviralAgents .
(3)  ?drug3 drugbank:drugCategory dbcategory:antihypertensiveAgents .
(4)  ?I1 drugbank:interactionDrug2 ?drug1 .
(5)  ?I1 drugbank:interactionDrug1 ?drug .
(6)  ?I2 drugbank:interactionDrug2 ?drug2 .
(7)  ?I2 drugbank:interactionDrug1 ?drug .
(8)  ?I3 drugbank:interactionDrug2 ?drug3 .
(9)  ?I3 drugbank:interactionDrug1 ?drug .
(10) ?drug drugbank:keggCompoundId ?cpd .
```

Drugbank

```
(11) ?enzyme kegg:xSubstrate ?cpd .
(12) ?enzyme rdf:type kegg:Enzyme .
(13) ?reaction kegg:xEnzyme ?enzyme .
(14) ?reaction kegg:equation ?equation .
```

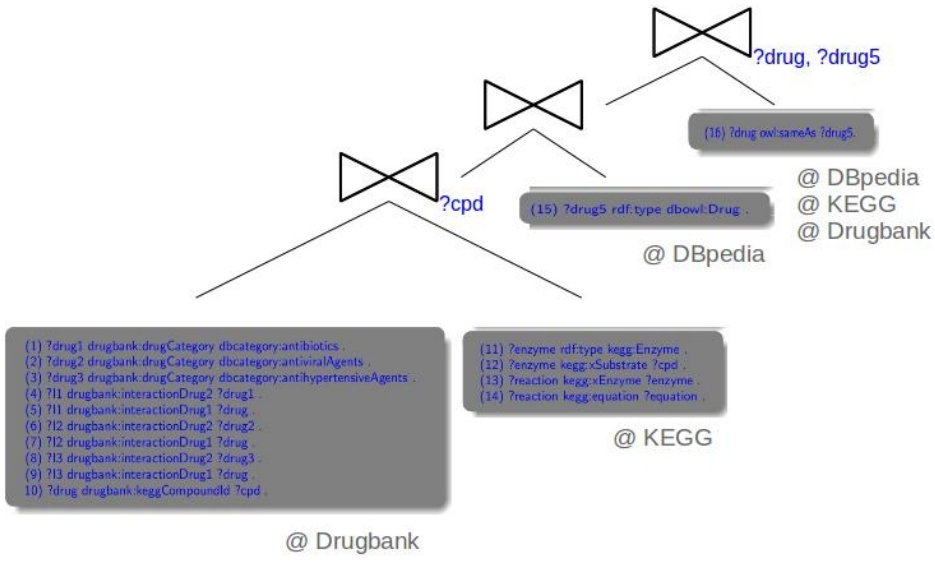
KEGG

```
(15) ?drug5 rdf:type dbowl:Drug .
```

DBpedia

```
(16) ?drug owl:sameAs ?drug5 .
```

DBpedia, KEGG, Drugbank



Execution plan

Adaptivity at Query Execution Level Coarse-Grained Granularity

- SPARQL-DQP [6] and ARQ [1] exploit information on SPARQL 1.1 queries to decide where the subqueries of triple patterns will be executed. Both rely on statistics or heuristics during query optimization, reaching thus coarse-grained adaptation at plan level.
- FedX [24] no adaptation is performed during query processing.

Finer granularity is achieved whenever the statistics are re-computed frequently.

[1] http://jena.sourceforge.net/ARQ

Adaptivity at Query Execution Level Fine-Grained Granularity

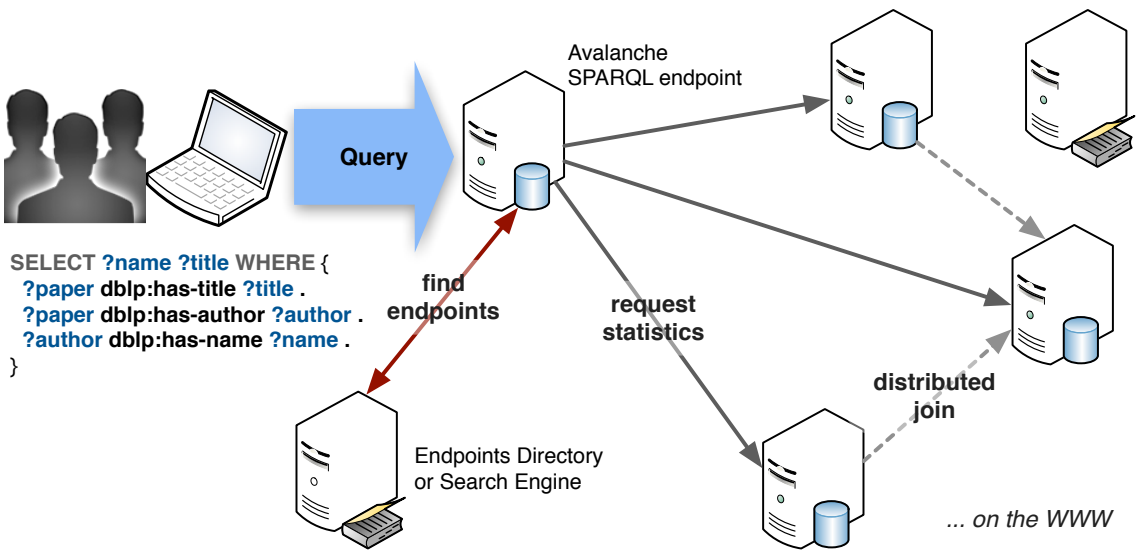
AVALANCHE [5]

- inter-operator solution to heuristically select on-the-fly the most promising plans.
- statistics about cardinalities and data distribution are considered to identify possibly good plans.
- competition strategy top-k plans are executed in parallel and the output of a query corresponds to the answers produced by the most promising plan(s) in a given period of time.

Fine-grained granularity. Decision of completely executing top-k plan(s) is taken on-the-fly, based on the current conditions of the environment.

Adaptivity at Query Execution Level Fine-Grained Granularity

Avalanche – execution model



Borrows from:

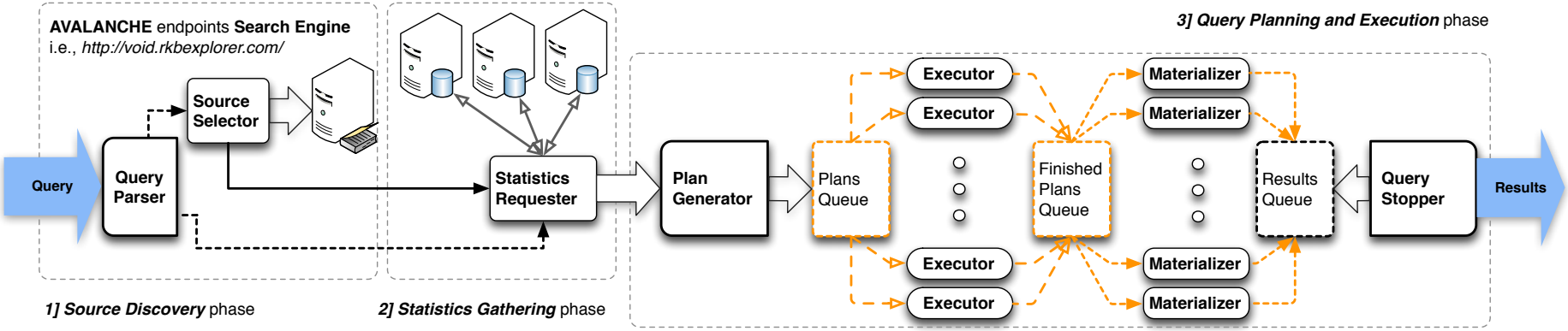
- p2p systems
- the Web

No completeness guarantees, *first K* results are obtained iteratively (plan-level) given *stopping criteria*:

- timeout
- relative saturation
- `LIMIT` modifier

Adaptivity at Query Execution Level Fine-Grained Granularity

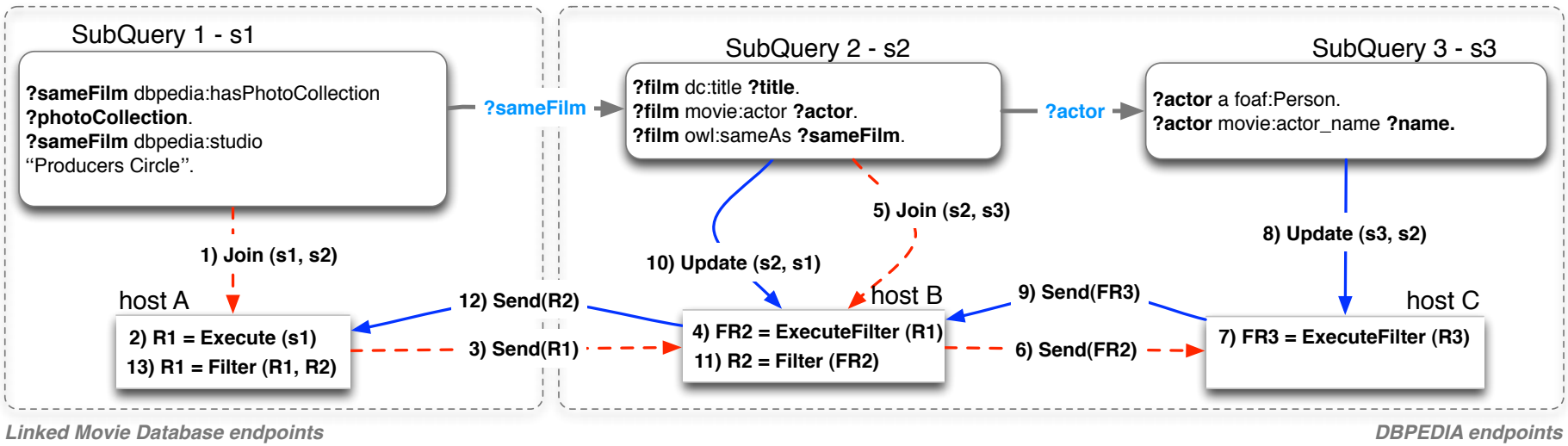
Avalanche – architecture



- optimization is concomitant with query execution
- scheduling is dynamic between plans (highlighted) – results come in *out-of-order*
- plan space: $|Hosts| * |SubQueries| * PlanDepth$
- multi-path depth-first heuristic search based on parametric cost model

Adaptivity at Query Execution Level Fine-Grained Granularity

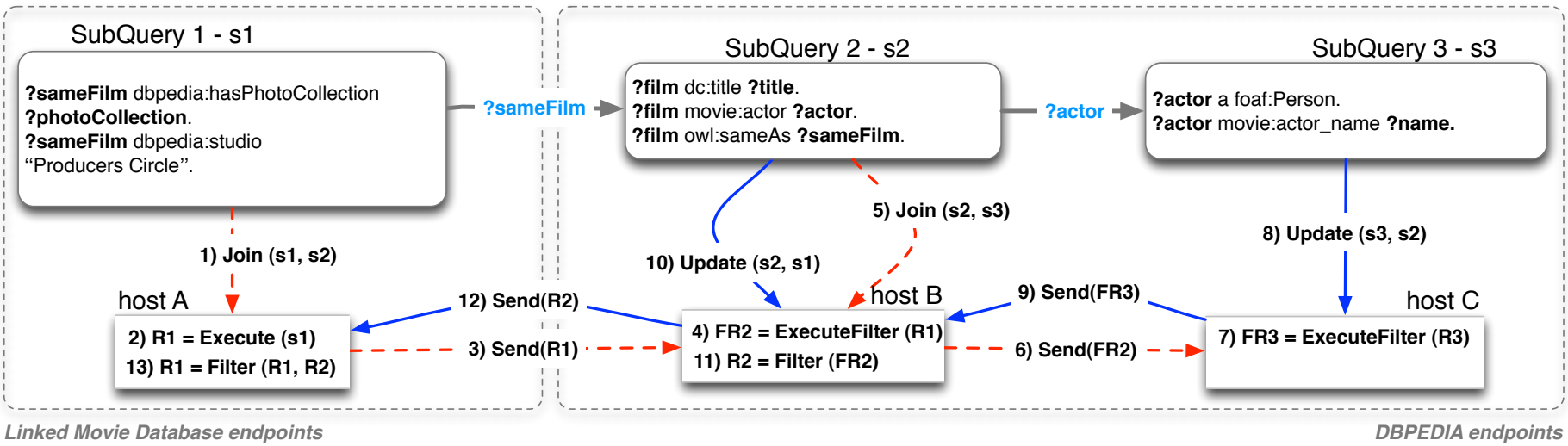
Avalanche – plan execution



- left linear execution strategy
- not adaptive

Adaptivity at Query Execution Level Fine-Grained Granularity

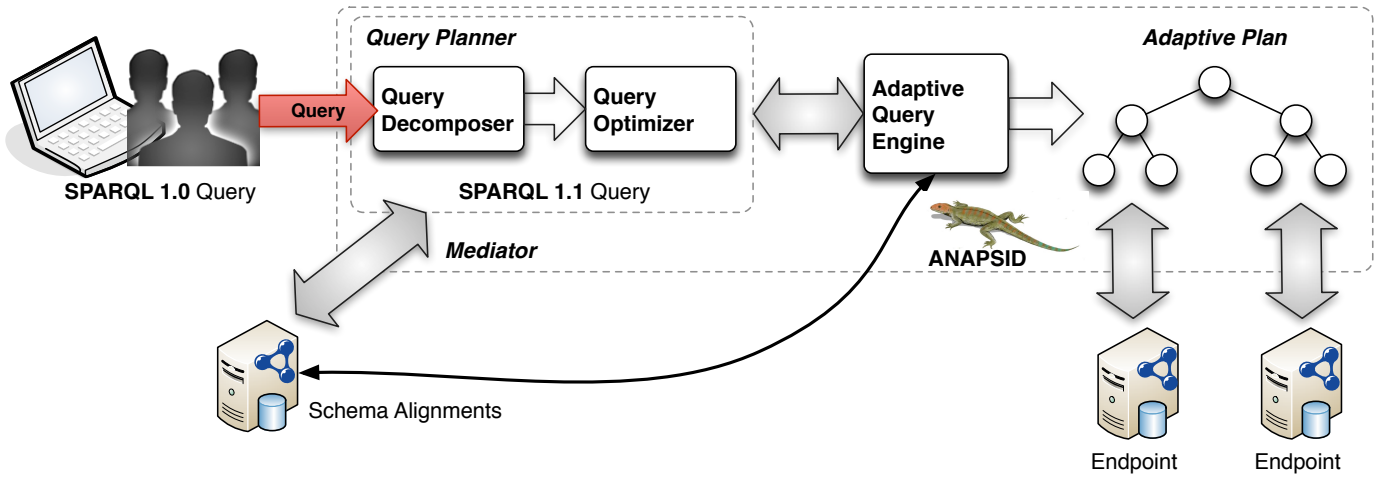
Avalanche – plan execution



- left linear execution strategy
- not adaptive
- could benefit from adaptive execution . . .

ANAPSID: An Adaptive Query Processing Engine for Federation of Endpoints [1]

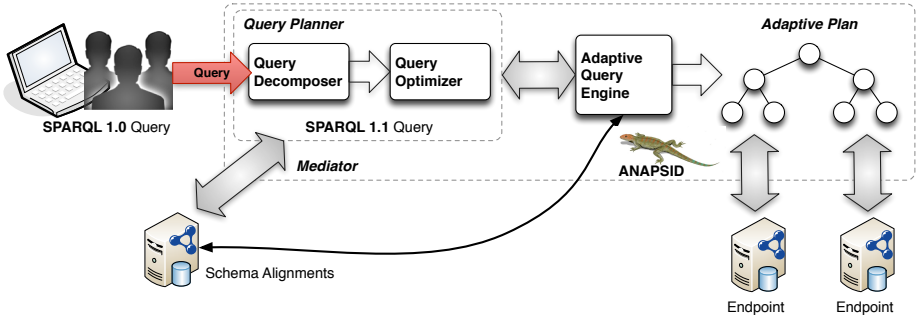
ANAPSID – Architecture



- Decomposer: rewrites the query into sub-queries.
- Optimizer: generates an optimal bushy tree execution plan.
- Query Engine: implements inter- and intra-operators techniques to perform fine-grained adaptivity.

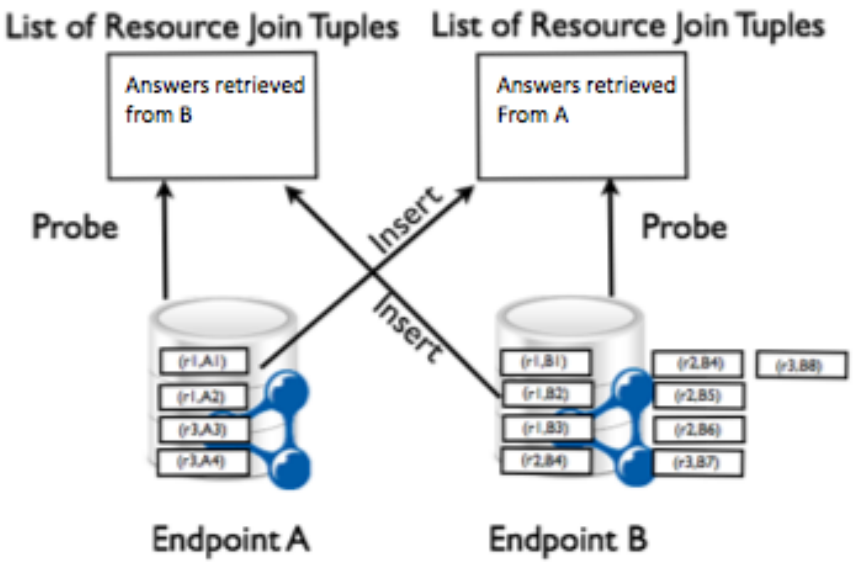
ANAPSID: An Adaptive Query Processing Engine for Federation of Endpoints [1]

- Adaptivity is performed at both source selection and query execution levels.
- Endpoint descriptions used to select endpoints that can answer a triple pattern; triple patterns that can be executed by the same endpoint are grouped together; endpoints may be contacted on-the-fly to decide if it is able to answer a triple pattern.
- Bushy-tree plans are generated to reduce the size of intermediate results and the number of Cartesian products.
- Non-blocking operators efficiently retrieve data from endpoints and keep producing new answers even when endpoints become blocked; agJoin, agUnion and agOptional provide adaptive implementations of SPARQL logical operators.



Fine-grained granularity during query execution and medium fine-grained during endpoint selection.

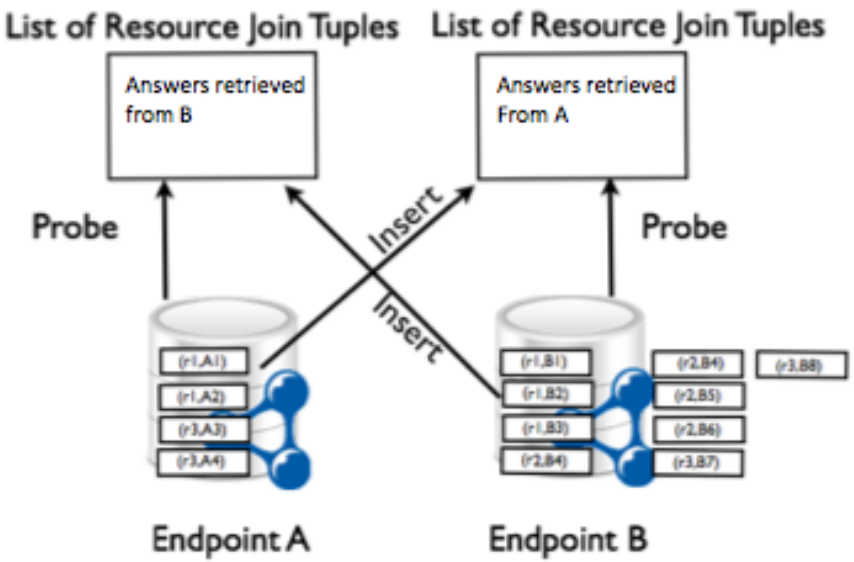
ANAPSID Adaptive Join Operator



- Operators implement main memory replacement policies to flush previously computed matches to secondary memory, ensuring no duplicate generation.
- Each operator maintains a data structure called Resource Join Tuple (RJT), that records for each instantiation of the variable(s), the tuples that have already matched.

$$(R, \{TR_1, TR_2, ..., TR_n\}).$$

ANAPSID Adaptive Join Operator



Adaptive Join Operator works in three stages:

- First stage is performed while at least one endpoint sends data. If main memory becomes full, Resource Join Tuples are flushed to secondary memory.
- Second stage is performed whenever both are blocked. Resource Join Tuples is main memory are joined with Resource Join Tuples in secondary memory
- Third stage is fired when both endpoints finish sending data.

Fedbench Datasets and Queries

Dataset	Endpoint name	#triples
NY Times	News	314k
LinkedMDB	Movies	6.14M
Jamendo	Music	1.04M
Geonames	Geography	7.98M
SW Dog Food	SW	84k
KEGG	Chemicals	10.9M
Drugbank	Drugs	517k
ChEBI	Compounds	4.77M
SP2B-10M	Bibliographic	10M
DBPedia subset	Infobox_Types	5.49M
	Infobox_Properties	10.80M
	Titles	7.33M
	Articles_Categories	10.91M
	Images	3.88M
	SKOS_Categories	2.24M
	Other	2.45M

Queries

Cross Domain CD1-CD7.

Linked Data LD1-LD11.

Life Science Linked Data LSD1-LD7.

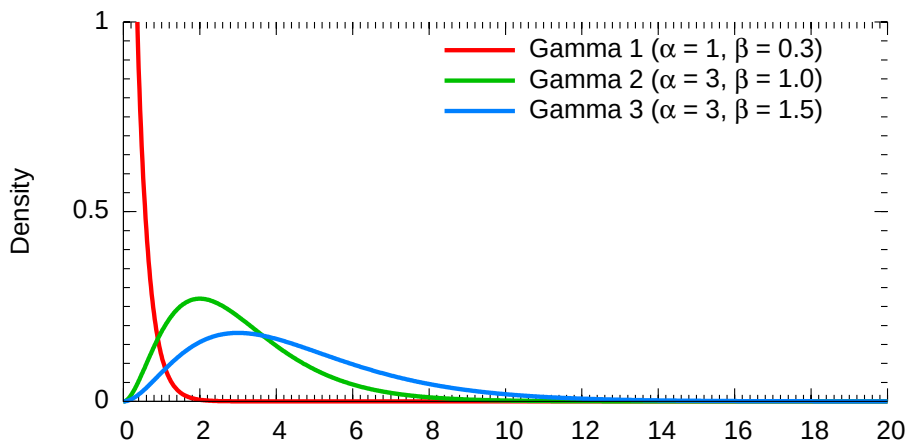
Complex Queries C1-C5.

Delays Configuration

Message Size: 16KB

- **No Delay**: **perfect network**.
- **Gamma 1** : **fast network** with a gamma distribution ($\alpha = 1$, $\beta = 0.3$) of response latency resulting in an average latency of 0.3 seconds.
- **Gamma 2**: **medium fast network** with a gamma distribution ($\alpha = 3$, $\beta = 1.0$) of response latency resulting in an average latency of 3 seconds.
- **Gamma 3**: **slow network** with a gamma distribution ($\alpha = 3$, $\beta = 1.5$) of response latency resulting in an average latency of 4.5 seconds per message.

Probability Density Function for Gamma



Evaluation Parameters

		Exec Time	Result Completeness
Query	# patterns	✓	✓
	instantiations	✓	✓
	result size	✓	
Data	data distribution	✓	✓
	data correlations	✓	✓
	dataset size	✓	
Network	#endpoints	✓	✓
	latency	✓	
	data transfer distribution	✓	

Evaluation

Objective: Study the Adaptive Capability of the Different Engines to Network Variation.

Measurements:

- **Time first tuple:** elapsed time between the execution of the plan and the output of the first answer.
- **Time all tuples:** elapsed time between the execution of the plan and the output of the whole answer.
- **Number of Results:** number of tuples produced for each query.

Summary and Future Directions

[1] Maribel Acosta, Maria-Esther Vidal, Tomas Lampo, Julio Castillo, and Edna Ruckhaus. ANAPSID: AN Adaptive query ProcesSing engIne for sparql enDpoints. In *Proceedings of the International Semantic Web Conference (ISWC)*, 2011.

[2] Gennady Antoshenkov and Mohamed Ziauddin. Query processing and optimization in oracle rdb. *VLDB J.*, 5(4):229–237, 1996.

[3] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *SIGMOD Conference*, pages 261–272, 2000.

[4] Shivnath Babu and Pedro Bizarro. Adaptive query processing in the looking glass. In *CIDR*, pages 238–249, 2005.

[5] Cosmin Basca and Abraham Bernstein. Avalanche: Putting the Spirit of the Web back into Semantic Web Querying. In *SSWS2010 Workshop*, Shanghai, China, 2010.

[6] Carlos Buil-Aranda, Marcelo Arenas, and Oscar Corcho. Semantics and optimization of the sparql 1.1 federation extension. In *ESWC (2)*, pages 1–15, 2011.

[7] Richard L. Cole and Goetz Graefe. Optimization of dynamic query evaluation plans. In *SIGMOD Conference*, pages 150–160, 1994.

[8] Amol Deshpande and Joseph M. Hellerstein. Lifting the burden of history from adaptive query processing. In *VLDB*, pages 948–959, 2004.

[9] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.

[10] Olaf Görlitz and Steffen Staab. SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions. In *Proceedings of the 2nd International Workshop on Consuming Linked Data*, Bonn, Germany, 2011.

[11] Goetz Graefe. Query evaluation techniques for large databases. *ACM Comput. Surv.*, 25(2):73–170, 1993.

[12] 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.

[13] 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.

[14] Olaf Hartig. Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In *Extended Semantic Web Conference*, 2011.

[15] 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.

[16] Yannis E. Ioannidis. Query optimization. *ACM Comput. Surv.*, 28(1):121–123, 1996.

[17] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *SIGMOD Conference*, pages 106–117, 1998.

[18] Kamlesh Laddhad and S. Sudarshan. Adaptive query processing. Technical Report 05329014, Kanwal Rekhi School of Information Technology, Indian Institute of Technology, Bombay, Mumbai, 2006.

[19] Gunter Ladwig and Thanh Tran. Linked data query processing strategies. In *International Semantic Web Conference (ISWC)*, 2010.

[20] Yingjie Li and Jeff Heflin. Using reformulation trees to optimize queries over distributed heterogeneous sources. In *ISWC*, pages 502–517, 2010.

[21] 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.

[22] Priti Mishra and Margaret H. Eich. Join processing in relational databases. *ACM Comput. Surv.*, 24(1):63–113, 1992.

[23] Bastian Quilitz and Ulf Leser. Querying distributed rdf data sources with sparql. In *ESWC*, pages 524–538, 2008.

[24] Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt. Fedx: Optimization techniques for federated query processing on linked data. In *International Semantic Web Conference (1)*, pages 601–616, 2011.

[25] Feng Tian and David J. DeWitt. Tuple routing strategies for distributed eddies. In *VLDB*, pages 333–344, 2003.

[26] Tolga Urhan and Michael J. Franklin. Xjoin: A reactively-scheduled pipelined join operator. *IEEE Data Eng. Bull.*, 23(2):27–33, 2000.

[27] Tolga Urhan and Michael J. Franklin. Dynamic pipeline scheduling for improving interactive query performance. In *VLDB*, pages 501–510, 2001.

[28] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *SIGMOD Conference*, pages 130–141, 1998.

[29] Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *VLDB*, pages 285–296, 2003.

[30] Eugene Wong and Karel Youssefi. Decomposition-a strategy for query processing. *ACM Trans. on Database Systems*, 1(3), 1976.