Runtime Query Optimization

Robust Optimization for Graphs

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

RDF Join Order Optimization

- Typical approach
  - Assign estimated *cardinality* to each triple pattern.
    - Bigdata uses the fast range counts
  - Start with the most selective triple pattern
  - For each remaining join
    - Propagate variable bindings
    - Re-estimate cardinality of remaining joins
- Cardinality estimation error
  - Grows *exponentially* with join path length

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Runtime Query Optimizer (RTO)

- Inspired by ROX (Runtime Optimization for XQuery)
  - Online, incremental and adaptive dynamic programming.
  - Breadth first estimation of actual cost of join paths.
  - Cost function is cumulative intermediate cardinality.
  - Extensions for SPARQL semantics.
- Query plans optimized in the *data* for each *query.*
  - Never produces a bad query plan.
  - Does not rely on statistical summaries.
  - Recognizes correlations in the data for the specific query.
  - Can improve the running time for some queries by 10x - 1000x.
  - Maximum payoff for high volume analytic queries.

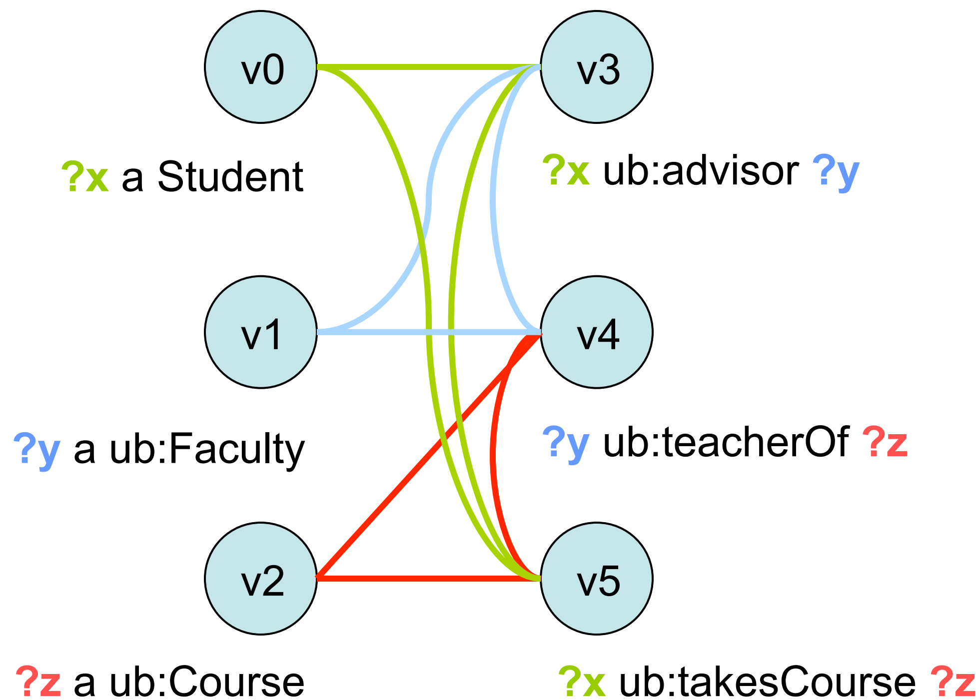
© 2006-2014 All Rights Reserved

Join Graphs

- Model set of joins as vertices
  - Connected by join variables
  - Implicit connections if variables shared through filters.
- Requires an index for each join
  - Easily satisfied for RDF.
- Starting vertices
  - Determined by their range counts.

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Join Graph for LUBM Q9

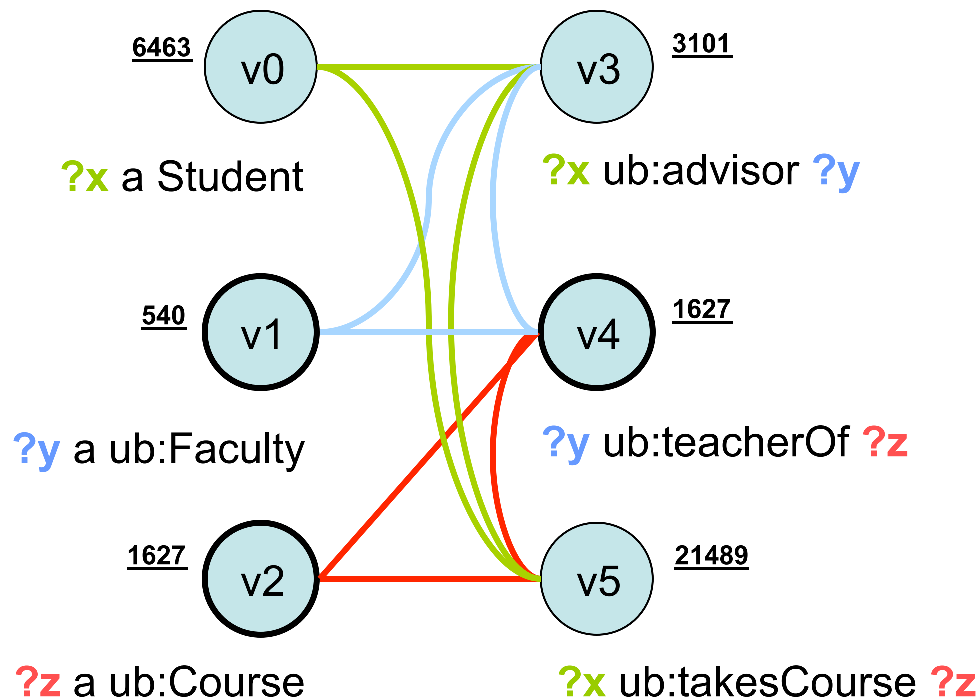



```
SELECT ?x ?y ?z
WHERE {
  ?x a ub:Student .          # v0
  ?y a ub:Faculty .          # v1
  ?z a ub:Course .           # v2
  ?x ub:advisor ?y .         # v3
  ?y ub:teacherOf ?z .       # v4
  ?x ub:takesCourse ?z . # v5
}
```

- 6 Vertices
- 3 Variables, each in 3 vertices.
- No filters.

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Annotate with Range Counts



- Range count each vertex.
- Minimum cardinality for {1,4,2}.
- Will begin with those vertices.

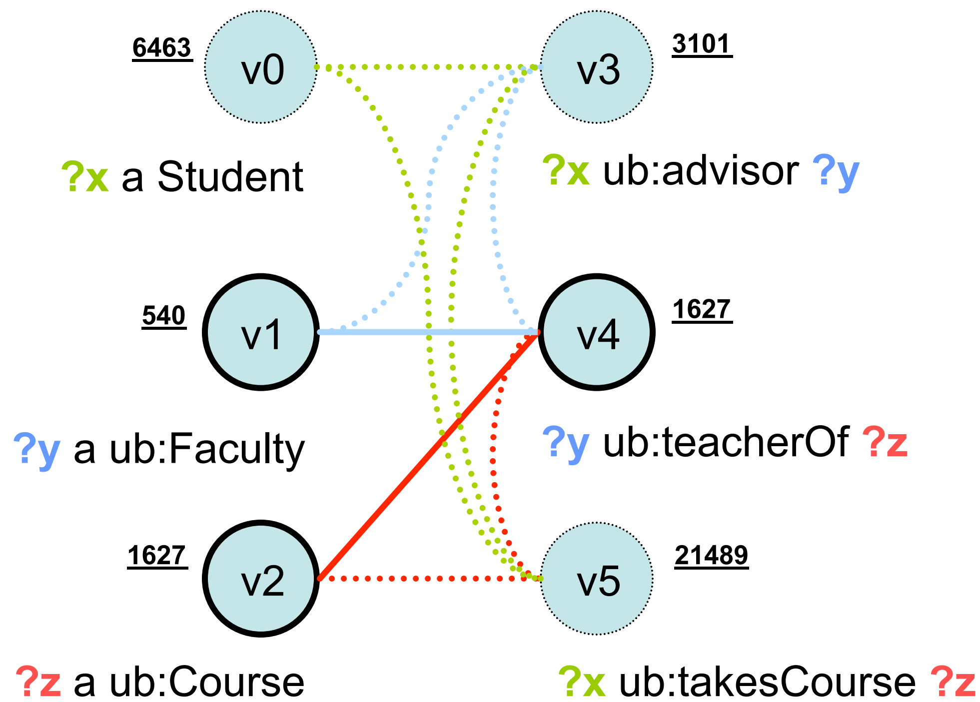
SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Incremental Join Path Estimation

- Evaluation proceeds in rounds.
- Increase sample size in each round.
  - Resample to reduce sampling bias.
  - Estimation error reduced as sample size increases.
- Estimate join cardinality using *cut off* joins
  - Push random sample of *N* tuples into each vertex
  - Join "cut off" when *N* tuples are output (handles overflow).
  - Output cardinality estimate can *underflow* (raise *N* in next round).
- Join paths pruned when dominated by another path for the same vertices.

© 2006-2014 All Rights Reserved

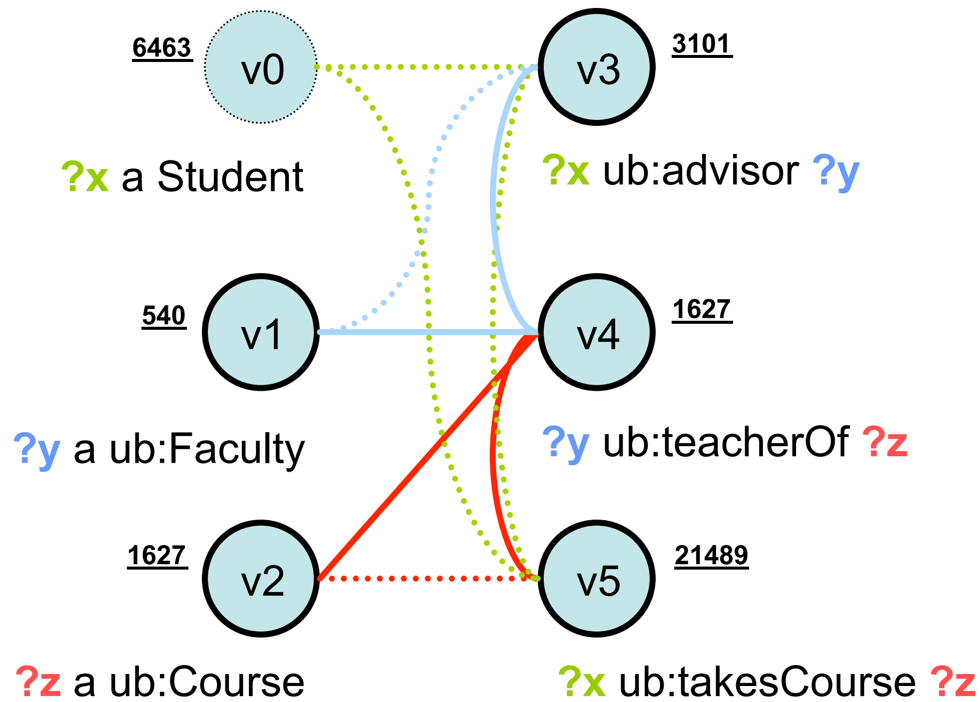
Round 0



Path : sumEstCard
- [1 4] : 1400
- [4 2] : 1627

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

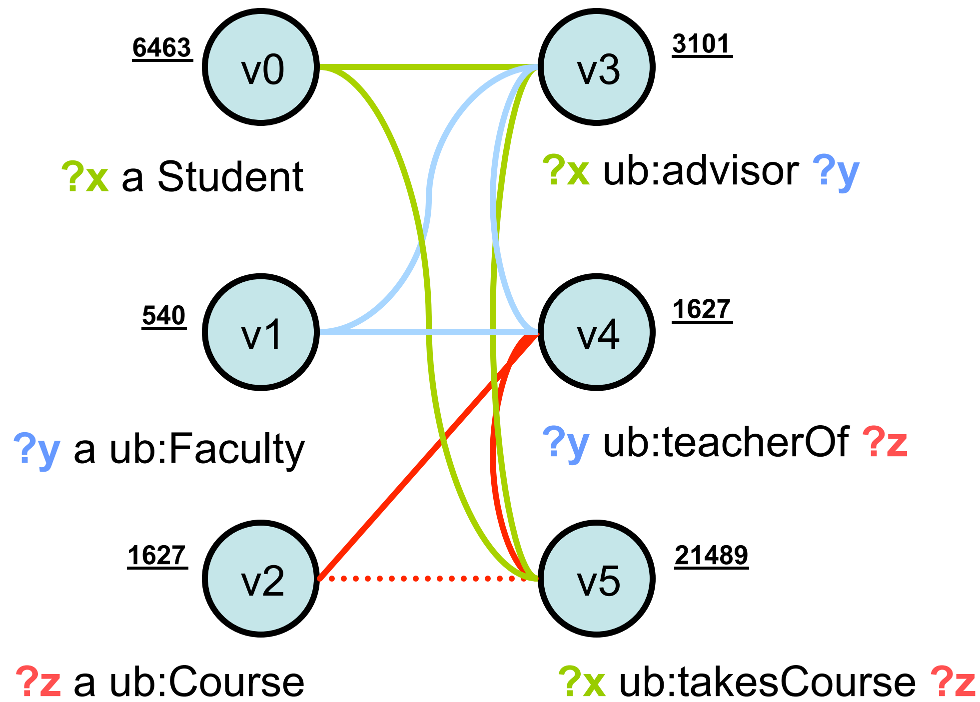
Round 1



Path : sumEstCard
- [1 4 2] : 1500
- [1 4 3] : 1500
- [1 4 5] : 1500
- [4 2 1] : 3254
- [4 2 3] : 10421
- [4 2 5] : 21964

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

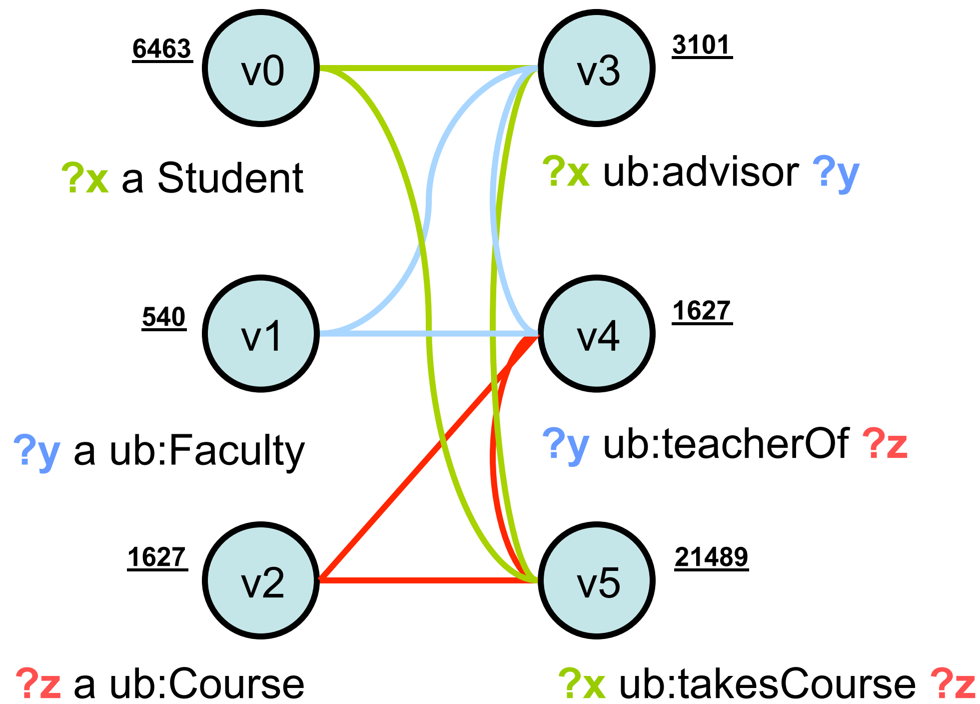
Round 2



Path : sumEstCard
- [1 4 3 0] : 8373
- [1 4 3 5] : 1686
- [1 4 5 0] : 44166
- [4 2 1 3] : 12685
- [4 2 1 5] : 30370
- [4 2 3 0] : 15639
- [4 2 3 5] : 10578
- [4 2 5 0] : 49080

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

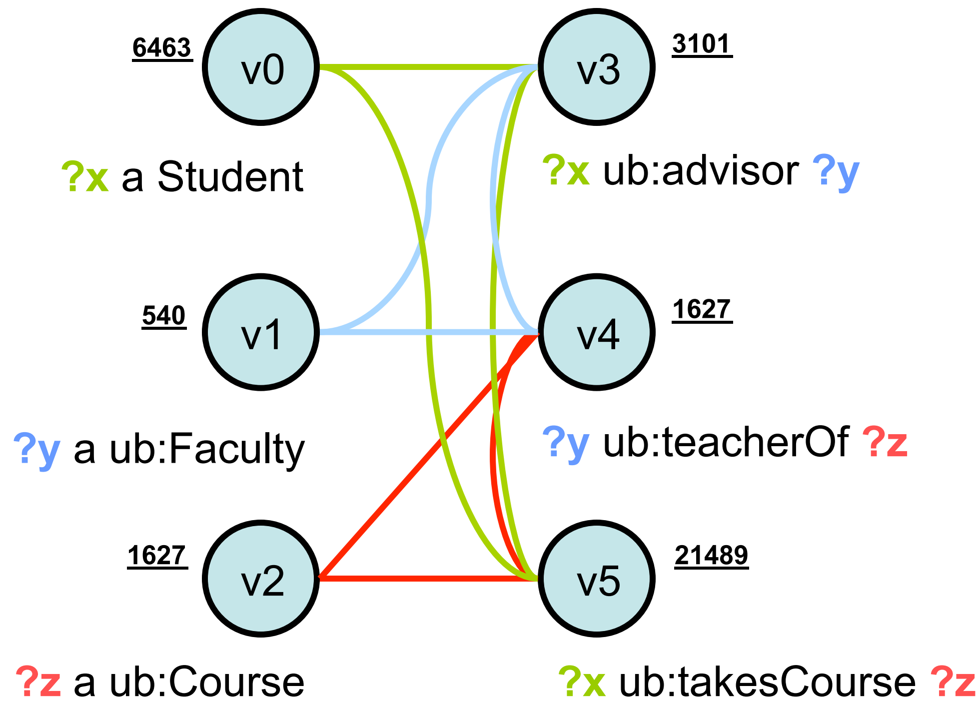
Round 3



Path : sumEstCard
- [1 4 3 0 2] : 14440
- [1 4 3 5 0] : 1791
- [1 4 3 5 2] : 1857
- [4 2 1 5 0] : 71045
- [4 2 3 5 0] : 10704

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

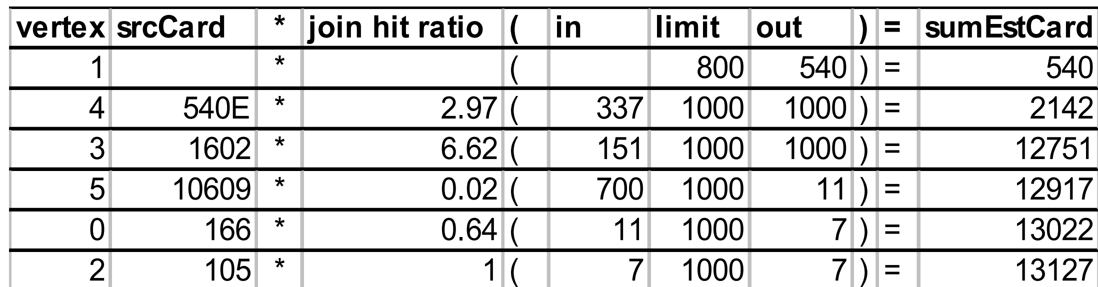
Round 4



Path : sumEstCard
- [1 4 3 5 0 2] : 1896

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Selected join path



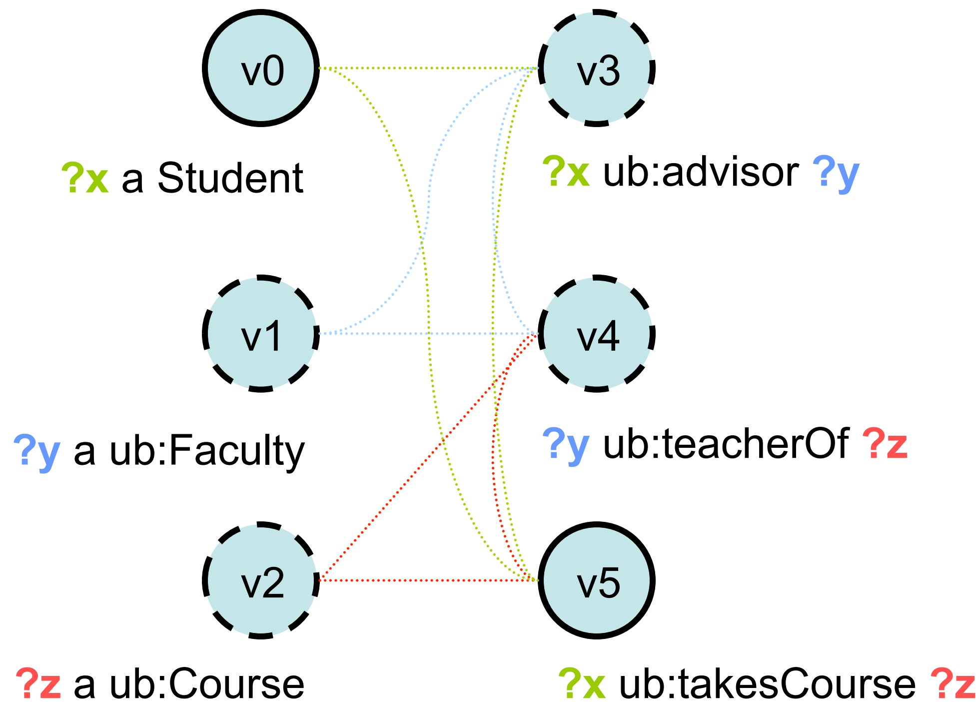
vertex	srcCard	*	join hit ratio	(	in	limit	out	)	=	sumEstCard
1		*		(		800	540	)	=	540
4	540E	*	2.97	(	337	1000	1000	)	=	2142
3	1602	*	6.62	(	151	1000	1000	)	=	12751
5	10609	*	0.02	(	700	1000	11	)	=	12917
0	166	*	0.64	(	11	1000	7	)	=	13022
2	105	*	1	(	7	1000	7	)	=	13127

75% Faster than the join path chosen by the static join order optimizer.

`vertex` : The vertices in the join path in the selected evaluation order.
`srcCard` : The estimated input cardinality to each join (E means *Exact*).
`ratio` : The estimated join hit ratio for each join.
`in` : The #of *input* solutions for each cutoff join.
`limit` : The sample size for each cutoff join.
`out` : The #of *output* solutions for each cutoff join.
`sumEstCard` : The cumulative estimated cardinality of the join path.

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Sub-Groups, Sub-Select, etc.



- Vectored left-to-right evaluation
- Solutions flowing into a group depend on join order in the parent

```
SELECT ?x ?y ?z WHERE {
  ?y a ub:Faculty .              # v1
  ?z a ub:Course .               # v2
  ?x ub:advisor ?y .             # v3
  ?y ub:teacherOf ?z .           # v4
  {    ?x a ub:Student .         # v0
       ?x ub:takesCourse ?z .    # v5
} }
```

- Flatten into join graph
- Path extensions MUST respect SPARQL group boundaries
- Sample join path with groups:

  1 => 4 => 3 { => 5 => 0 } => 2

SYSTAP™, LLC
© 2006-2014 All Rights Reserved

Bigdata® Vectored Query Engine

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Bigdata Query Engine

- Vectored query engine
  - High concurrency and vectored operator evaluation.
- Physical query plan (BOPs)
  - Supports pipelined, blocked, and at-once operators.
  - Chunks queue up for each operator.
  - Chunks transparently *mapped* across a cluster.
- Query engine instance runs on:
  - Each data service; *plus*
  - Each node providing a SPARQL endpoint.

© 2006-2012 All Rights Reserved

Query Hints

- Virtual triples
  - Scope to query, graph pattern group, or join.
- Allow fine grained control over
  - Analytic query mode
  - Evaluation order
  - Vector size
  - Etc.

```
SELECT ?x ?y ?z
WHERE {
      hint:Query hint:vectorSize 10000 .
      ?x a ub:GraduateStudent .
      ?y a ub:University .
      ?z a ub:Department .
      ?x ub:memberOf ?z .
      ?z ub:subOrganizationOf ?y .
      ?x ub:undergraduateDegreeFrom ?y .
}
```

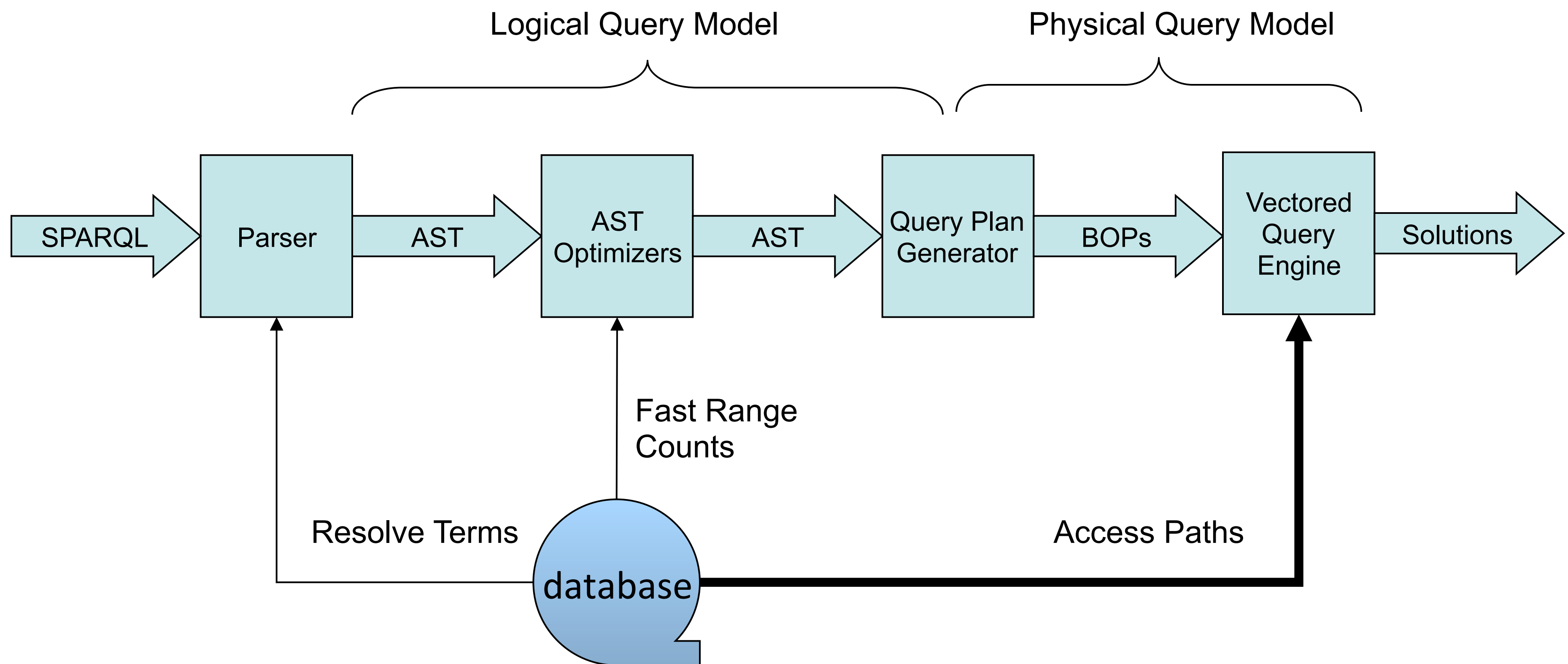

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

New Bigdata Join Operators

- Pipeline joins (fast, incremental).
  - Map *binding sets* over the shards, executing joins close to the data.
  - Faster for single machine and *much* faster for distributed query.
  - First results appear with very low latency
- Hash joins against an access path
  - Scan the access path, probing the hash table for joins.
  - Hash joins on a cluster can saturate the disk transfer rate!
- Solution set hash joins
  - Combine sets of solutions on shared join variables.
  - Used for Sub-Select and Group Graph Patterns (aka join groups)
- Merge joins
  - Used when a series of group graph patterns share a common join variable
    - Think OPTIONALs.
  - Java : Must sort the solution sets first, then a single pass to do the join.
  - HTree : *Linear* in the data (it does not need to sort the solutions).

© 2006-2012 All Rights Reserved

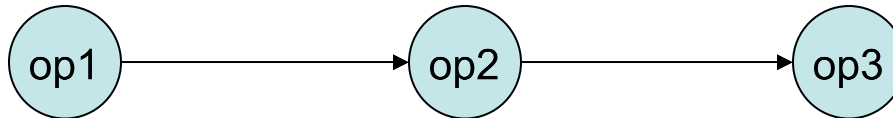
Query Evaluation



SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Data Flow Evaluation

- Left-to-right evaluation
  - Physical query plan is operator sequence (not tree)

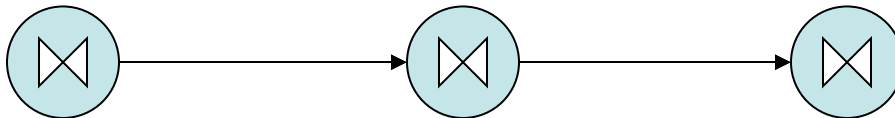


- Parallelism
  - Within operator
  - Across operators
  - Across queries
- Operator annotations used extensively.

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

"Pipeline" Joins

- Fast, vectored, indexed join.
- Very low latency to first solutions.



- Mapped across shards on a cluster.
  - Joins execute close to the data.
- Faster for single machine
  - *Much* faster for distributed query.

© 2006-2012 All Rights Reserved

Preparing a query

Original query:

```
SELECT ?x WHERE {
?x a ub:GraduateStudent ;
   ub:takesCourse <http://www.Department0.University0.edu/
   GraduateCourse0>.
}
```

Translated query:

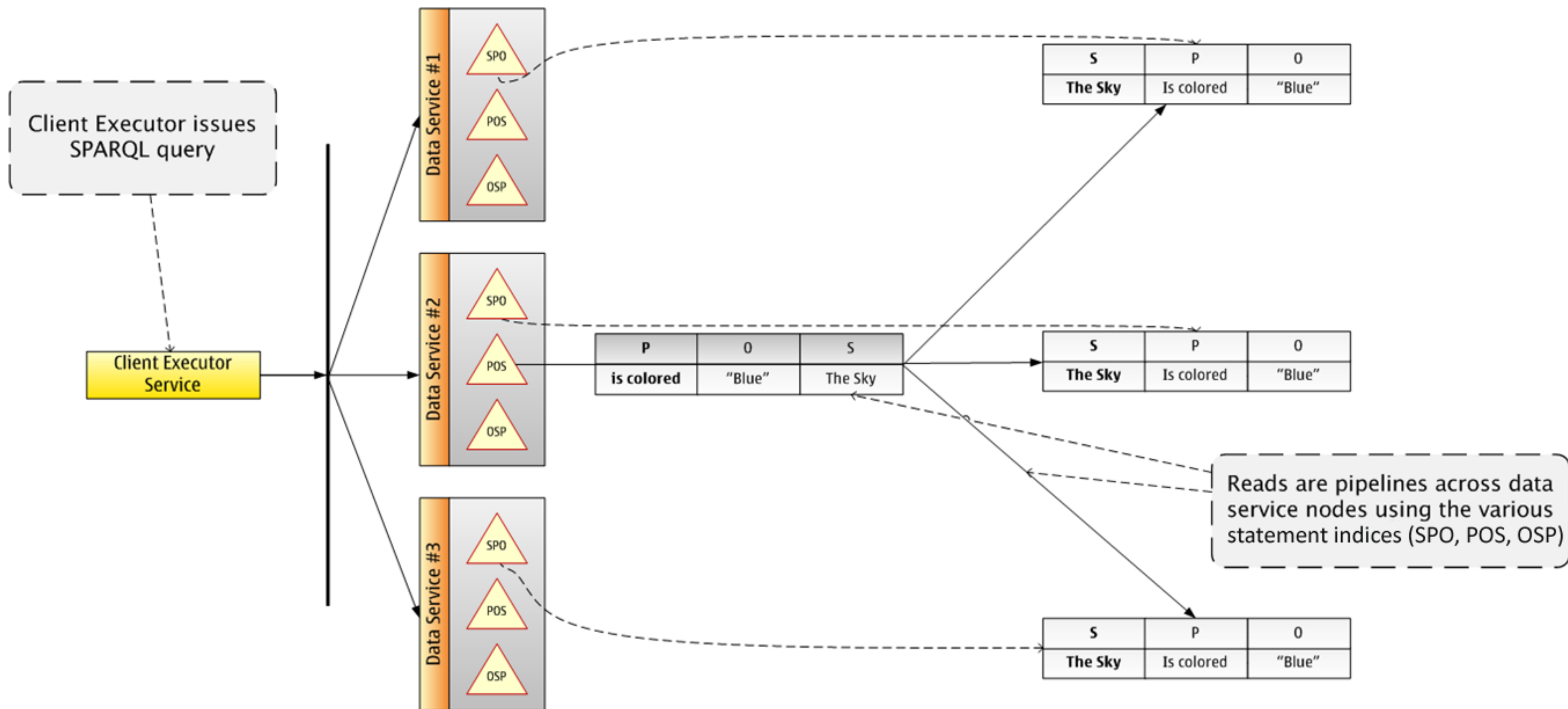
```
query :- (x 8 256) ^ (x 400 3048)
```

Query execution plan (access paths selected, joins reordered):

```
query :- pos(x 400 3048) ^ spo(x 8 256)
```


SYSTAP™, LLC
© 2006-2012 All Rights Reserved

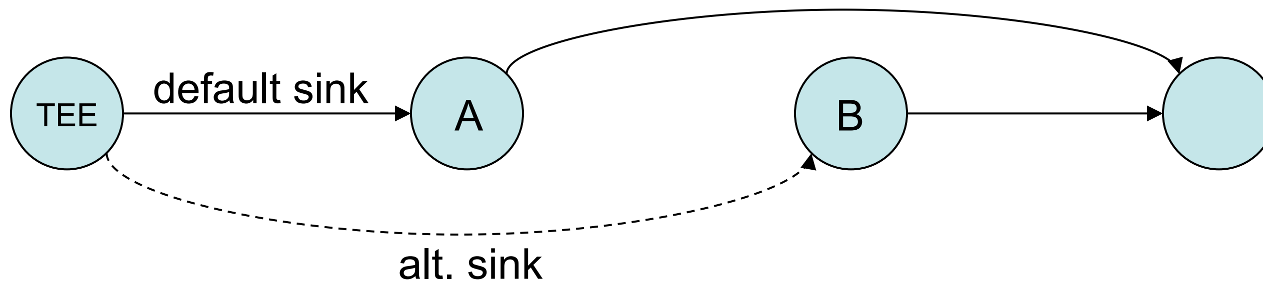
Pipeline Join Execution



SYSTAP™, LLC
© 2006-2012 All Rights Reserved

TEE

- Solutions flow to both the default and alt sinks.
- UNION(A,B)

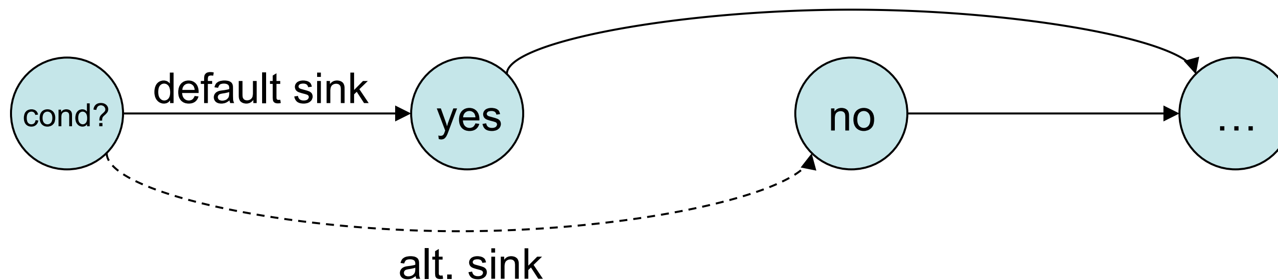


- Generalizes to n-ary UNION

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Conditional Flow

- Data flows to either the default sink or the alt sink, depending on the condition.

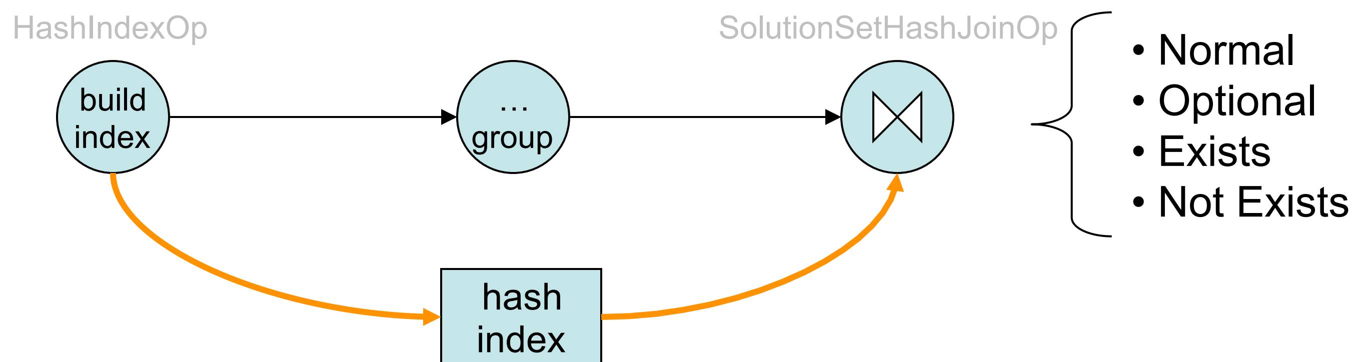


- Branching pattern depends on the query logic.

© 2006-2012 All Rights Reserved

Sub-Group, Sub-Select, etc.

- Build a hash index from the incoming solutions (at-once).
- Run the sub-select, sub-group, etc.
  - Solutions from the hash index are vectored into the sub-plan.
  - Sub-Select must hide variables which are not projected.
- Join against the hash index (at-once).



SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Named Solution Sets

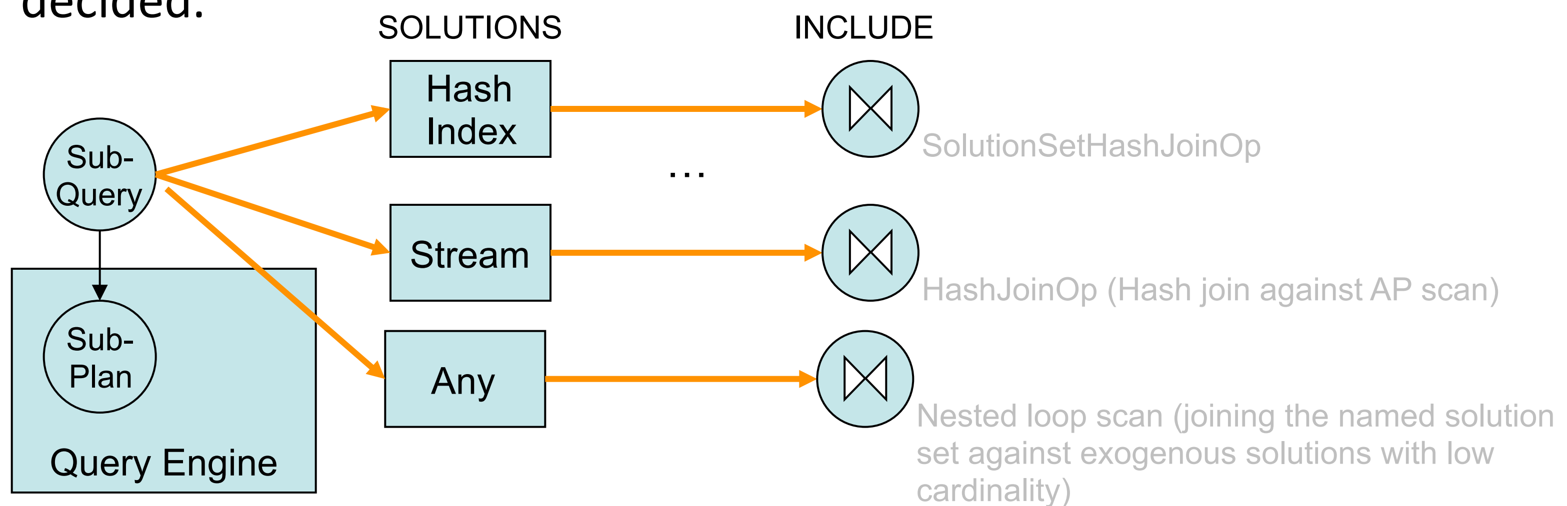
- Support solution set reuse
- Evaluation
  - Run before main WHERE clause.
  - Exogenous solutions fed into named subquery.
  - Results written on hash index.
  - Hash index INCLUDEd into query.
- Example is BSBM BI Q5

```
Select ?country ?product ?nrOfReviews ?avgPrice
WITH {
    Select ?country ?product (count(?review) As ?nrOfReviews)
    {
      ?product a <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/
     v01/instances/ProductType4> .
      ?review bsbm:reviewFor ?product .
      ?review rev:reviewer ?reviewer .
      ?reviewer bsbm:country ?country .
    }
    Group By ?country ?product
} AS %namedSet1
WHERE {
  { Select ?country (max(?nrOfReviews) As ?maxReviews)
    {
       INCLUDE %namedSet1
    }
    Group By ?country
  }
  { Select ?product (avg(xsd:float(str(?price))) As ?avgPrice)
    {
      ?product a <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/
     v01/instances/ProductType4> .
      ?offer bsbm:product ?product .
      ?offer bsbm:price ?price .
    }
    Group By ?product
  }
  INCLUDE %namedSet1 .
  FILTER(?nrOfReviews=?maxReviews)
}
Order By desc(?nrOfReviews) ?country ?product
```


SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Named Subquery Evaluation

- Run the sub-query on the query engine.
- Output written onto hash index or stream.
  - Hash index is a good choice if the join variables can be identified in advance, but that is often not possible since the position of the INCLUDE(s) for that solution set in the rest of the query are not yet decided.



SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Bottom Up Evaluation

- SPARQL semantics specify "as if" bottom up evaluation
  - For many queries, left-to-right evaluation produces the same results.
  - Badly designed left joins must be lifted out and run first.

```
SELECT * {
:x1 :p ?v .
OPTIONAL {
  :x3 :q ?w .
  OPTIONAL { :x2 :p ?v }
  }
}
```

```
SELECT * {
WITH {
  SELECT ?w ?v {
    :x3 :q ?w .
    OPTIONAL { :x2 :p ?v }
  }
} AS %namedSet1
:x1 :p ?v .
  OPTIONAL {
  INCLUDE %namedSet1
  }
}
```


SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Merge Join Pattern

- High performance n-way join
  - Linear in the data (on HTree).
- Series of group graph patterns must share a common join variable.
- Common pattern in SPARQL
  - Select something and then fill in optional information about it through optional join groups.

```
SELECT (SAMPLE(?var9) AS ?var1) ?var2 ?var3
{
    { SELECT DISTINCT ?var3
      WHERE {
          ?var3 rdf:type pol:Politician.
          ?var3 pol:hasRole ?var6.
          ?var6 pol:party "Democrat".
      }
    }
    OPTIONAL {
       ?var3 p1:name ?var9
    }
    OPTIONAL {
    ?var10 p2:votedBy ?var3.
    ?var10 rdfs:label ?var2.
    }
    ...
}
GROUP BY ?var2 ?var3
```


SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Merge Joins (cont.)

```
SELECT (SAMPLE(?var9) AS ?var1) ?var2 ?var3
{
    { SELECT DISTINCT ?var3
      WHERE {
            ?var3 rdf:type pol:Politician.
            ?var3 pol:hasRole ?var6.
            ?var6 pol:party "Democrat".
      }
    }
    OPTIONAL {
       ?var3 p1:name ?var9
    }
    OPTIONAL {
    ?var10 p2:votedBy ?var3.
    ?var10 rdfs:label ?var2.
    }
    ...
}
GROUP BY ?var2 ?var3
```

SELECT …

WITH {… } as %set1

WITH {… } as %set2

WITH {… } as %set3

WHERE {

INCLUDE %set1.

OPTIONAL { INCLUDE %set2. }

OPTIONAL { INCLUDE %set3. }

}

Handled as one n-way merge join.

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Parallel Graph Query Plans

- Partly ordered scans
  - Parallel AP reads against multiple shards
- Hash partitioned
  - Distinct
  - Order by
  - Aggregation (when not pipelined)
- Default Graph Queries
  - Global index view provides RDF "Merge" semantics.
  - DISTINCT TRIPLES across the set of contexts in the "default graph"

© 2006-2012 All Rights Reserved

Parallel Hash Joins for Unselective APs

- AP must read a lot of data with reasonable locality
  - Shard view: journal + index segment(s).
    - One IO per leaf on segments.
    - 90%+ data in key-order on disk
    - Narrow stride (e.g., OSP or POS)
- Crossover once sequential IO dominates vectored nested index join
  - Key range scans turn into sequential IO in the cluster
  - Goal is to saturate the disk read bandwidth
- Strategy A
  - Map intermediate solutions against target shards
  - Build hash index on target nodes
  - Parallel local AP scans, probing hash index for joins.
- Strategy B
  - Hash partition intermediate solutions on join variables.
  - Parallel local AP scans against spanned shard(s)
    - Vectored hash partition of solutions read from APs.
    - Vectored probe in distributed hash index partitions for joins.

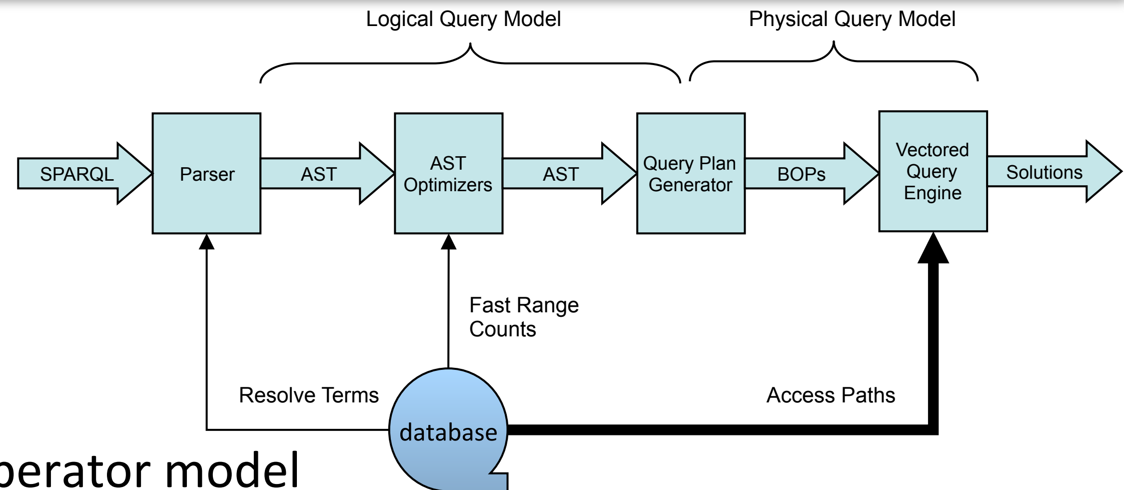
SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Examining a Query Plan

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

NSS "Explain"

- SPARQL Query
  - As given
- SPARQL Parse Tree
  - As parsed
- Original AST
  - Generated abstract operator model
- Optimized AST
  - Optimized abstract operator model
- Query Plan
  - Physical query plan
- Query Evaluation Statistics
  - As executed



SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Query Evaluation Statistics

- As-run or “live”
  - “Explain” provides “as-run” view.
  - Live view available by monitoring long running queries.
- Summary statistics
  - solutions=128, chunks=52, subqueries=0, elapsed=2467ms.
    - solutions #of generated solutions.
    - chunks #of “chunks” in the output solutions.
    - subqueries #of subqueries evaluated (each in their own table)
    - elapsed total elapsed clock time for this query.
- Followed by a detailed table of per-operator statistics.
  - For the top-level query
  - For each sub-query

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Detailed Operator Statistics (Many columns are for the cluster)

Column	Description
queryId	query or subquery identifier
evalOrder	*total* then one per operator in order.
bopId, predId	Index of operator or predicate in plan.
bopSummary	Short summary for each operator
predSummary	Short summary for access path (AP).
fastRangeCount	Estimated cardinality for AP.

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Units & Chunks In & Out

• unitsIn	• #of solutions flowing into an operator
• chunksIn	• #of chunks of solutions flowing into an op.
• unitsOut	• #of solutions flowing out of an operator.
• chunksOut	• #of chunks of solutions flowing out of an op.
• typeErrors	• #of runtime SPARQL type errors (solution dropped)
• joinRatio	• Ratio of solutions out to solutions in. – Indicator of selectivity and cardinality explosion.

SYSTAP™, LLC
© 2006-2012 All Rights Reserved

Query Engine Metrics
NSS / Ganglia

Work Queues – one per operator:

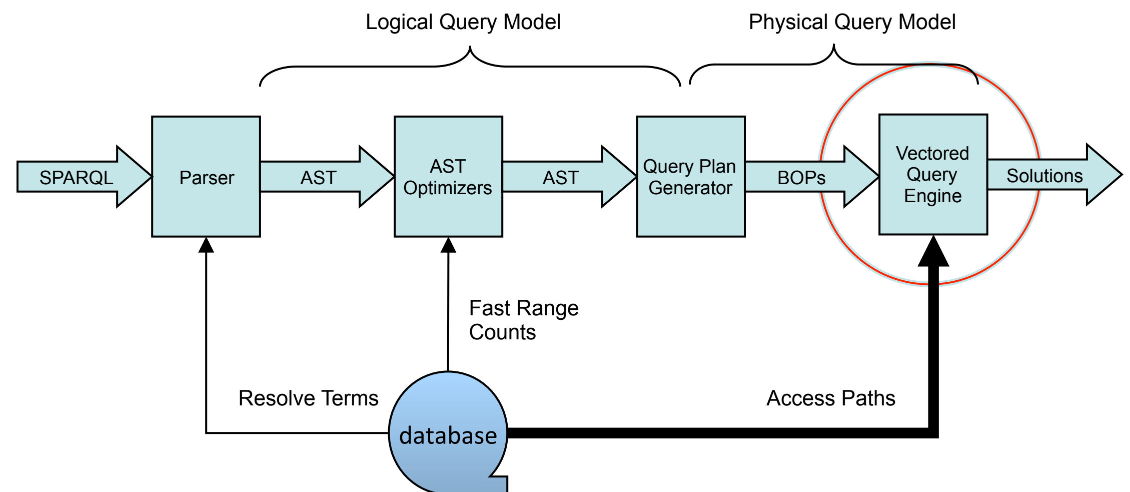
- blockedWorkQueueCount
- blockedWorkQueueRunningTotal

Operators – many per query plan:

- operatorActiveCount
- operatorHaltCount
- operatorStartCount
- operatorTasksPerQuery

Queries:

- queriesPerSecond
- queryDoneCount
- queryErrorCount
- queryStartCount



SYSTAP™, LLC
© 2006-2012 All Rights Reserved