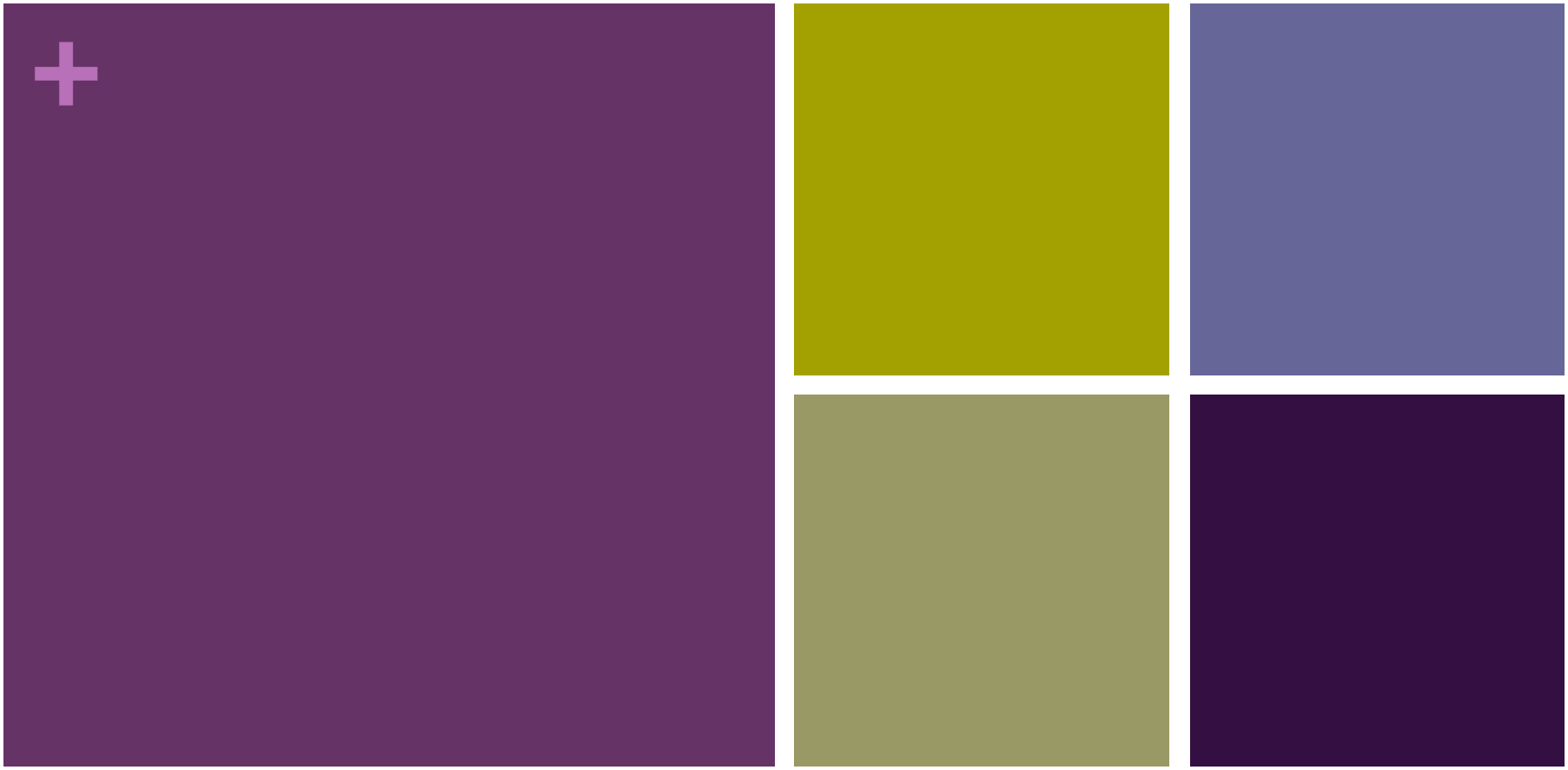

QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases

Jacopo Urbani <jacopo@cs.vu.nl> - Vrije Universiteit Amsterdam

QueryPIE in one slide

QueryPIE

- QueryPIE is a hybrid reasoning engine that can scale to a billion triples
- It greatly reduces the costs of backward-chaining by performing a small precomputation
  - Precomputation: 8-300sec against 1-3hours of WebPIE
  - Query response time: in the order of few milliseconds both with and without reasoning

QueryPIE in 17 slides

Reasoning on large data

- Rule-based reasoning =
  process that applies a set of rules recursively

- Example of rule:

  if <a type B>
  and <B subclassOf C>
  then <a type C>

- We can perform reasoning
  beforehand **(forward-chaining inference)**
  or at query time **(backward-chaining inference)**

Forward/Backward-chaining inference

- Forward-chaining reasoning
  - **Pros:**
    - Querying the data becomes a “database” problem
  - **Cons:**
    - Difficult to apply if data changes frequently
- Backward-chaining reasoning
  - **Pros:**
    - Reasoning performed only when necessary
  - **Cons:**
    - Even for small queries, it requires expensive reasoning

QueryPIE

- Currently, the state of the art is forward-chaining
- Forward-chaining methods (one for all is WebPIE) can scale to very large extend
- However, the intrinsic disadvantages of forward-chaining still remain!
- **QueryPIE:**
  - Combine the advantages of both approaches through hybrid reasoning
  - Get closer to reasoning that is service-oriented and more “webby”

QueryPIE

Hybrid reasoning

- The input of our method is a generic triple pattern (e.g. <?s ?p ?o>)
- In QueryPIE we apply forward-chaining reasoning only to derive the *schema (or terminological)* triples. All the other triples are derived at query-time
- Terminological triples:
  - Definition: triples that have a RDFS/OWL term as predicate and/or object and are used in the inference rules
  - e.g:
    - (?x rdfs:subClassOf ?y)
    - (?x rdfs:subPropertyOf ?y)
    - (?x rdf:type owl:TransitiveProperty)
    - etc.

Hybrid reasoning

- Why does it make sense to pre-compute the closure on the schema?
  - Schema does not change often
  - It is relatively small compared to the rest of the data
  - It is frequently used in the reasoning rules
- We define two algorithms:
  - *terminological-closure*
    (executed before query-time)
  - *terminology-independent reasoning*
    (executed during query-time)

Hybrid reasoning terminology-independent reasoning

Terminology-independent reasoning

- implements classical backward-chaining algorithm without performing inference on the schema
- some optimizations are possible only because the schema is pre-computed

  1) prune the reasoning tree by using the inferred schema (very important)

  2) replicate the schema and perform most of the joins where the data is

Hybrid reasoning terminological-closure

Terminological-closure

- **Problem**: cannot perform a partial closure in a forward-chaining fashion
- **Solution**: Exploit the terminology-independent reasoning to achieve this goal
- **Proposed algorithm**:

  1) Assume that the schema inference is already computed

  2) Query the engine asking for the schema

  3) Add the inferred triples to the input and repeat point 2 until fix point

In other words we perform forward reasoning with "backward steps"

Hybrid reasoning Implementation

What about the implementation of QueryPIE?

- Java distributed prototype that uses the Ibis framework for the node communication
- Triples are indexed with 4 indexes (*spo, sop, pos, ops*) and partitioned across a set of nodes
- Algorithm:
  - 1) One node receives in input a generic triple pattern (?x ?y ?z)
  - 2) This node generates a reasoning tree that might derive new data and send it to the nodes where the data is
  - 3) The data is collected into end location and returned to the user

Evaluation

Hybrid reasoning Evaluation

- Our goal was to compare the performance against the current state of the art without external overhead
  - **Full materialization scenario**:
    calculated full materialization (with WebPIE) and
    loaded the data in QueryPIE *deactivating* reasoning
  - **Hybrid reasoning scenario**:
    loaded data in QueryPIE and calculated the partial closure and perform reasoning at query-time
- Used the same hardware infrastructure for both scenario's:
  8 machines, each 2 CPUs and 24G main memory
- Considered 3 datasets:
  - LDSR (860M triples),
  - LLD (700M triples)
  - LUBM (1.1B triples)

Hybrid reasoning Evaluation

Terminological closure algorithm consistently reduces the complexity of pure backward-chaining:

Input pattern (LUBM pattern queries)	#leaves hybrid/backward-chaining	Ratio
?x ?y University	21/174	8.29
University0 hasAlumnus ?x	5/58	11.6
?x rdf:type ResearchGroup	2/3	1.5
UndergradStud0 rdf:type ?x	38/291	7.66

Hybrid reasoning Evaluation

- Terminological closure algorithm:
  Much faster than full materialization

Dataset	Terminological-closure	Full materialization (WebPIE)	Ratio
LUBM	8sec	1h15min	562
LLD	332sec	1h5min	11
FactForge (LDSR)	89sec	2h45min	111

Hybrid reasoning Evaluation

- terminology-independent algorithm: Overhead of reasoning not noticeable for the user

Query	# Results	Time w reas.	Time w/o reas.	Ratio
?x type CompactCar	182	3.38ms	3.32ms	1.02
?y ?x University	75613	55.12ms	35.75ms	1.54
?x uniprot:pathway… ?y	4	8.17ms	1.10ms	7.43
Arnold_Swart… ?x ?y	4937	8.57ms	2.86ms	2.99
?x rdf:type ResearchGroup	2400836	1166ms	1017ms	1.15

Take home message

Hybrid reasoning Conclusions

- on the fly inference can be done up to very large scale, using hybrid reasoning
- hybrid reasoning is more convenient than full-materialization in case data changes frequently
- QueryPIE is still work in progress
  - Move from triple patterns to SPARQL
  - Move from OWL Horst to OWL 2 RL