UNIVERSITY OF Southampton
School of Electronics and Computer Science

The Design and Implementation of Minimal RDFS Backward Reasoning in 4store

Manuel Salvadores, Gianluca Correndo, Steve Harris, Nick Gibbins, and Nigel Shadbolt

Contents

- Motivation
- Background
  - 4store
  - Minimal RDFS
- 4sr
  - Distributed Model
  - Design and Implementation
- LUBM Scalability Evaluation
- Conclusions

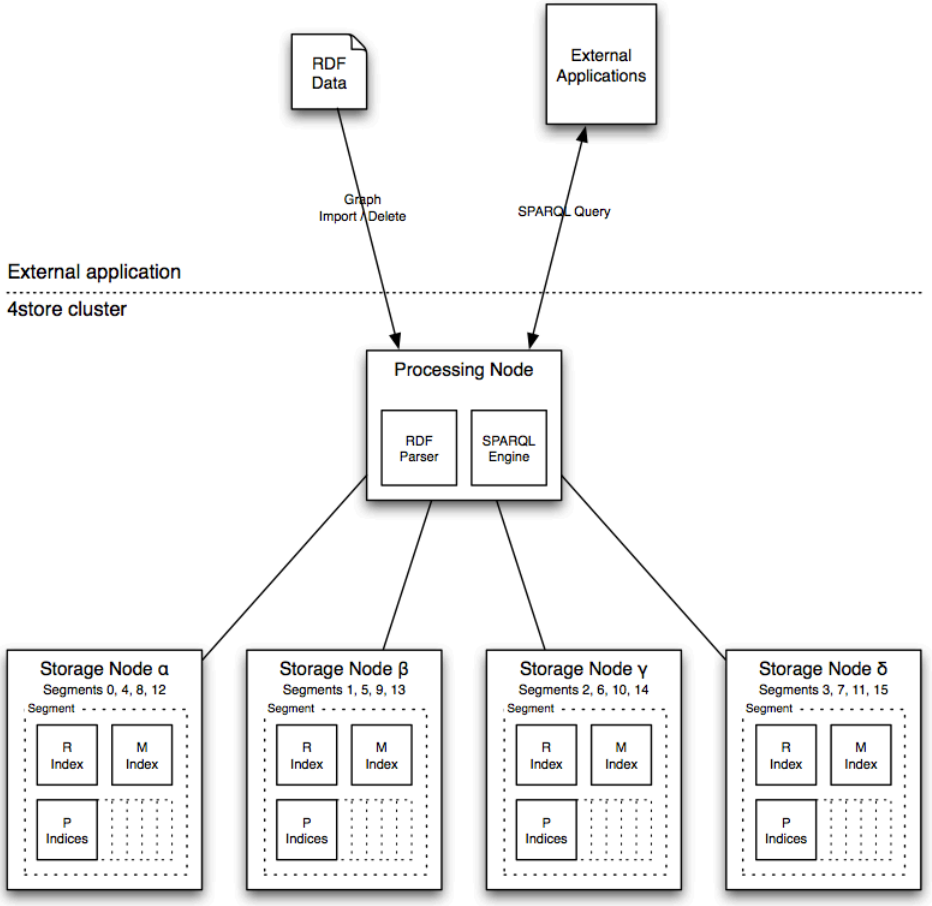
Motivation

- Triple/Quad stores are good for schema-less data engineering. Semantics in Triple/Quad stores are even better!
- Forward chained reasoning can be very expensive in space. Moreover, updates force to re-compute entailments.
- Data changes regularly and SPARQL/Update is in process of standardization ... we need to improve backward chained reasoning.

4store

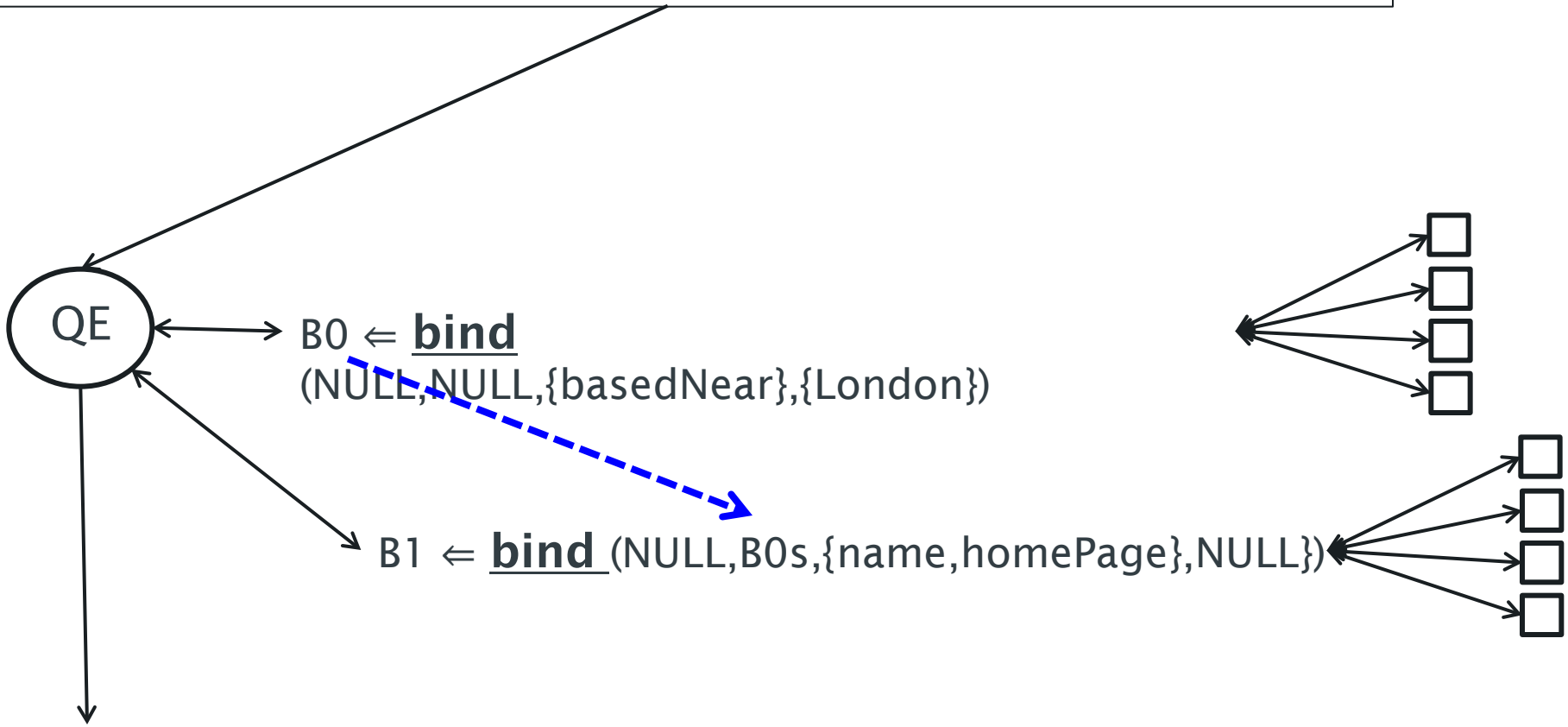
4store is a clustered RDF storage and SPARQL query system that became open source under the GNU license in July 2009.

- Clustered/Distributed (quads allocated on segment based on subject hash modulo)
- Written in C.
- Native storage (2 radix tries per predicate PO/PS, 1 hash for context)
- Native communication protocol on top of TCP/IP
- Fast, last LUBM Benchmark (2nd on import, 2nd on query and 1st on updates)



4store: bind operation

```
SELECT ?name ?page WHERE { ?x foaf:name ?name .
?x foaf:homePage ?page . ?x foaf:basedNear dbpedia:London }
```



Minimal RDFS

- Minimal RDFS refers to the RDFS fragment published in: Simple and Efficient Minimal RDFS Muñoz, S., Pérez, J., Gutierrez, C.:. Journal of Web Semantics 7, 220–234 (September 2009)
- RDFS Issues:
  - RDFS can generate inconsistencies.
  - Decidability issues.
  - No differentiation between language constructors and ontology vocabulary.
- Minimal RDFS is built upon the ρdf fragment which includes the following RDFS constructors: rdfs:subPropertyOf, rdfs:subClassOf, rdfs:domain, rdfs:range and rdf:type

4sr's Distributed Model

- Definitions
  - ρdf = {sc, sp, dom, range, type}
  - A quad (m,s,p,o) is an mrdf-quad iff p ∈ ρdf - {type}, and Gmrdf is a graph with all the mrdf-quads from every graph in a KB.

4sr's Distributed Model

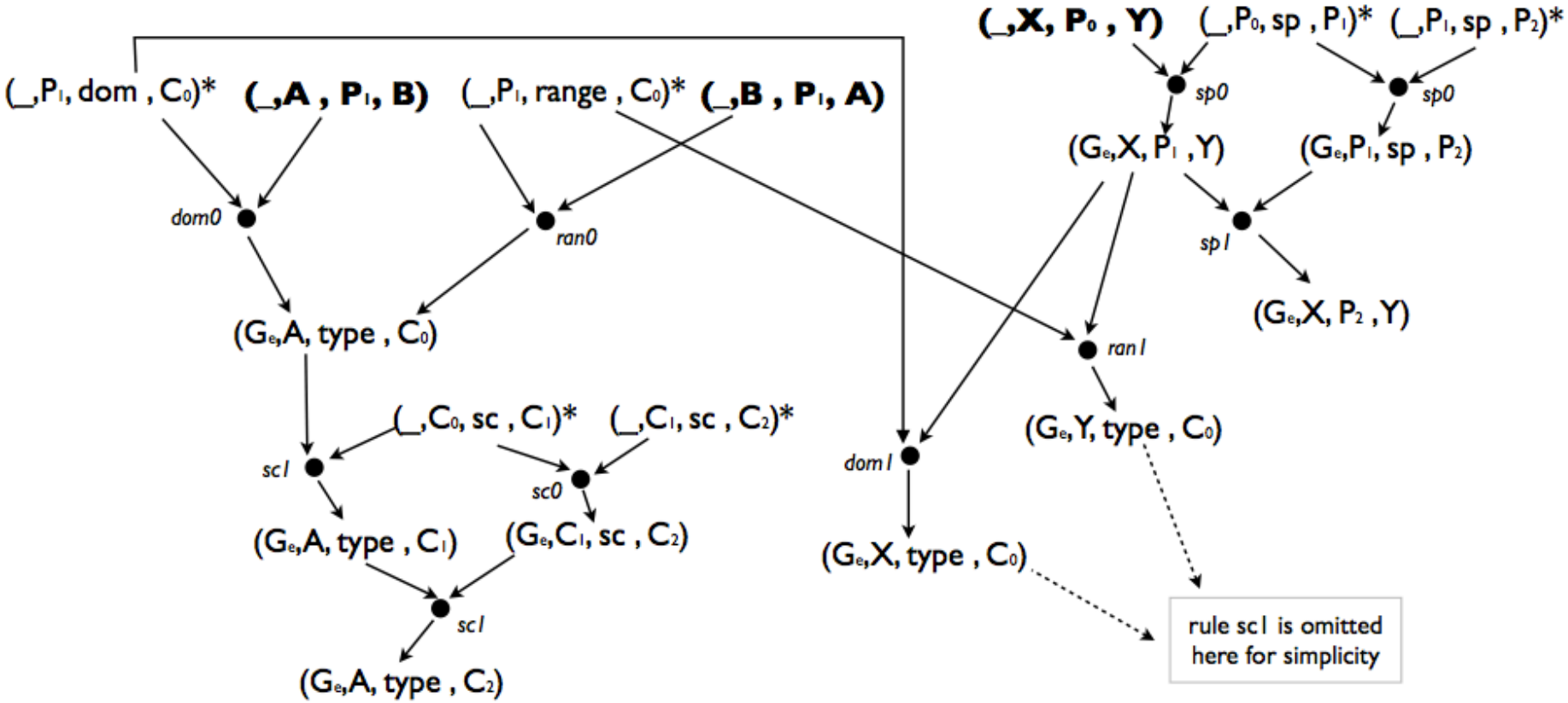
$$(sp_0)\ \frac{(\_, A, sp, B)(\_, B, sp, C)}{(G_e, A, sp, C)} \qquad (sp_1)\ \frac{(\_, A, sp, B)(\_, X, A, Y)}{(G_e, X, B, Y)}$$

$$(sc_0)\ \frac{(\_, A, sc, B)(\_, B, sc, C)}{(G_e, A, sc, C)} \qquad (sc_1)\ \frac{(\_, A, sc, B)(\_, X, type, A)}{(G_e, X, type, B)}$$

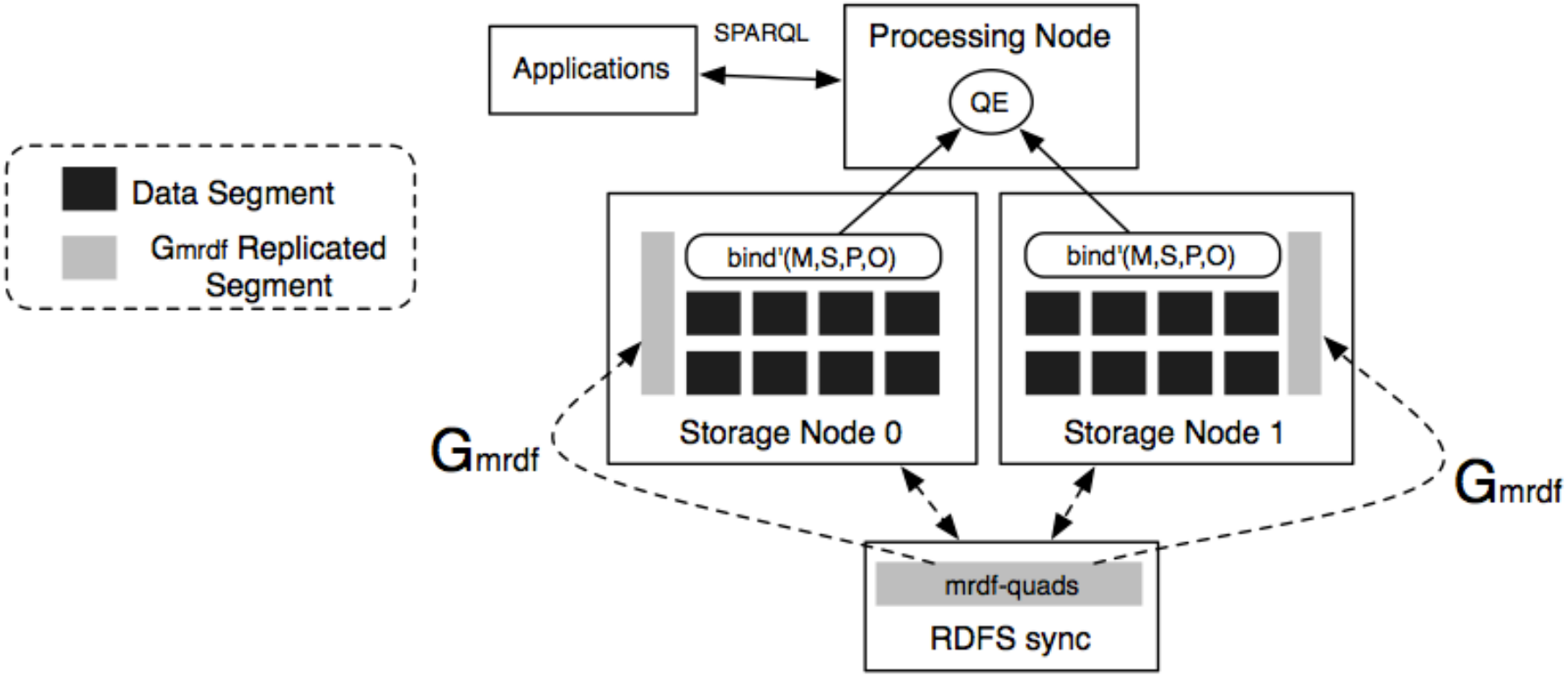
$$(dom_0)\ \frac{(\_, A, dom, B)(\_, X, A, Y)}{(G_e, X, type, B)} \qquad (ran_0)\ \frac{(\_, A, range, B)(\_, X, A, Y)}{(G_e, Y, type, B)}$$

$$(dom_1)\ \frac{(\_, A, dom, B)(\_, C, sp, A)(\_, X, C, Y)}{(G_e, X, type, B)} \qquad (ran_1)\ \frac{(\_, A, range, B)(\_, C, sp, A)(\_, X, C, Y)}{(G_e, X, type, B)}$$

4sr's Distributed Model



4sr's Design and Implementation

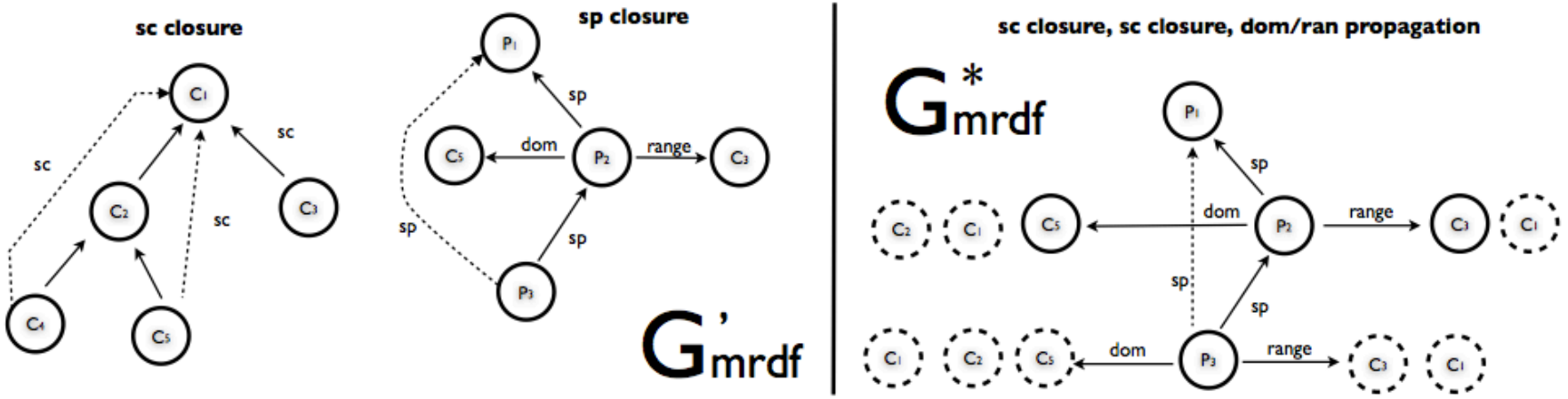


4sr's Design and Implementation

Simplified Algorithm of the original bind in 4store:
Input: B_M, B_S, B_P, B_O,segment **Output:** r - a list of quads

```
Let B* be the list of pattern combinations of B_M, B_S, B_P, B_O
For every pattern in B*
    Let T be the radix tree in segment with optimized iterator for pattern
    For every quad in T
        If (pattern_m = ∅ OR pattern_m = quad_m) AND
           (pattern_s = ∅ OR pattern_s = quad_s) AND
           (pattern_p = ∅ OR pattern_p = quad_p) AND
           (pattern_o = ∅ OR pattern_o = quad_o)
           append quad into r
```

4sr's Design and Implementation



- $G'_{mrdf|sc}(X)$ as the subclass closure of X in G'_{mrdf}.
- $G'_{mrdf|sp}(X)$ as the subproperty closure of X in G'_{mrdf}.
- $G^*_{mrdf|dom}(X)$ as the set of properties in G^*_{mrdf} with X as domain.
- $G^*_{mrdf|range}(X)$ as the set of properties in G^*_{mrdf} with X as range.
- $G^*_{mrdf|list}()$ the set of every inferable domain or range in G^*_{mrdf}.
- $G^*_{mrdf|bind}(s,p,o)$ the list of statements that is retrieved from a normal bind operation for a triple pattern (s,p,o).

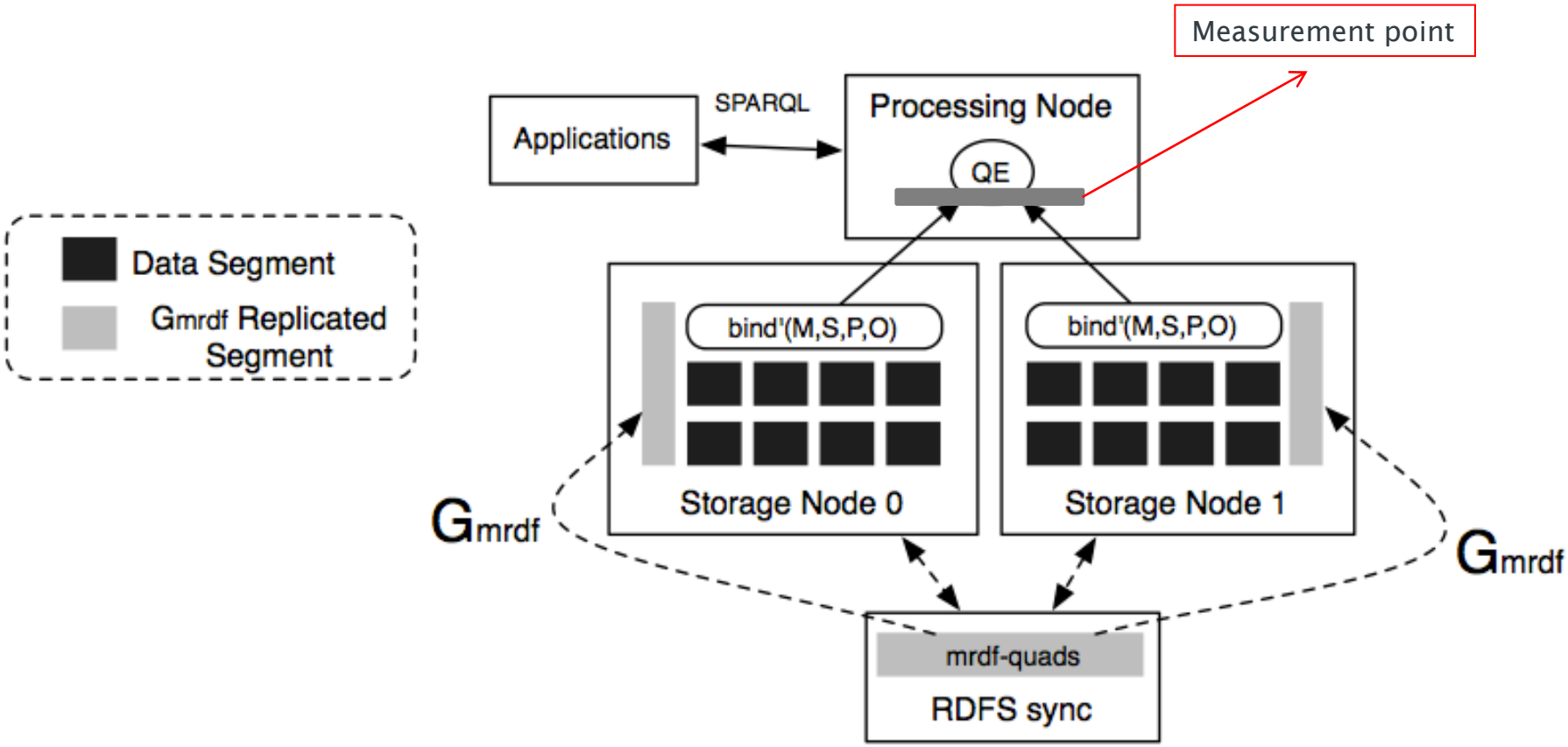
4sr's Design and Implementation

Bind' Algorithm in 4sr:
Input: B_M, B_S, B_P, B_O,segment **Output:** r - a list of quads

Let B^* be the list of pattern combinations of B_M, B_S, B_P, B_O
For every pattern in B^*
(a) If $|G'_{mrdf|sp}(pattern_p)| > 1$
    append to r bind($pattern_m$,$pattern_s$,$G'_{mrdf|sp}(pattern_p)$,$pattern_o$)
(b) Else If $pattern_p = \emptyset$
    For every pred in segment
        append to r bind'($pattern_m$,$pattern_s$,$pred$,$pattern_o$)
(c) Else If $pattern_p = type$
(c0) If $pattern_o \neq \emptyset$
        For s in bind($\emptyset$,$\emptyset$,$G^*_{mrdf|dom}(pattern_o)$,$\emptyset$)
            append to r (G_e,sol_s,$type$,$pattern_o$)
        For s in bind($\emptyset$,$\emptyset$,$G^*_{mrdf|ran}(pattern_o)$,$\emptyset$)
            append to r (G_e,sol_o,$type$,$pattern_o$)
        append to r bind($pattern_m$,$pattern_m$,$type$,$G'_{mrdf|sc}(pattern_o)$)
(c1) Else
        For object in $G^*_{mrdf|list}()$
            append to r bind'($pattern_m$,$pattern_s$,$type$,$object$)
(d) Else If $pattern_p \in (sc, sp, range, dom)$
    append to r $G^*_{mrdf|bind}(pattern_s, pattern_p, pattern_o)$
(e) Else
    append to r bind($pattern_m$,$pattern_s$,$pattern_p$,$pattern_o$)

LUBM Scalability Evaluation

- LUBM(100), LUBM(200), LUBM(400), ..., LUBM(1000).
- From 13M to 138M Triples.



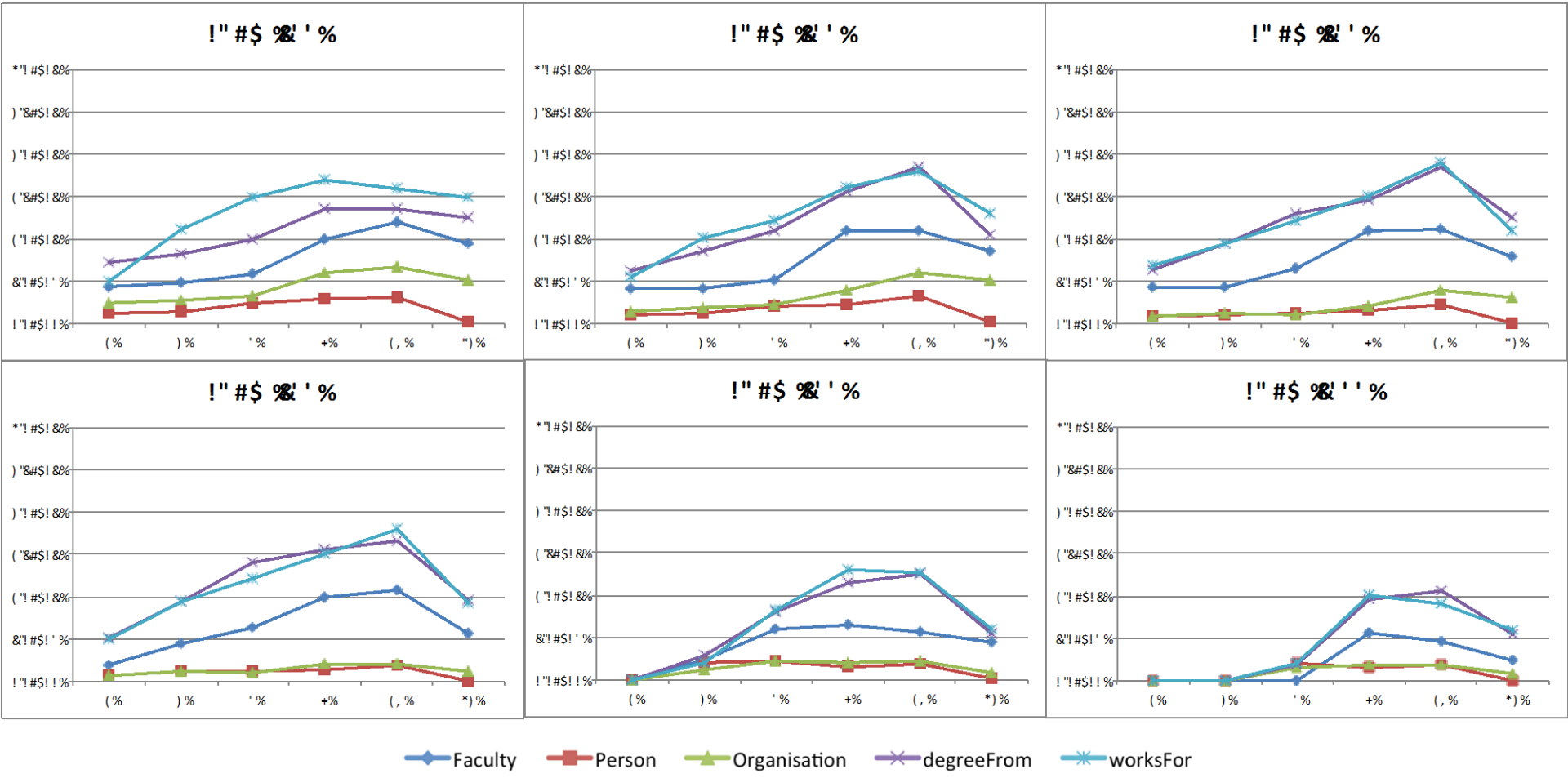
LUBM Scalability Evaluation

- Hardware Specs:
  - **Server set-up:** One Dell PowerEdge R410 with 2 dual quad processors (8 cores - 16 threads) at 2.40GHz, 48G memory and 15k rpm SATA disks.
  - **Cluster set-up:** An infrastructure made of 5 Dell PowerEdge R410s, each of them with 4 dual core processors at 2.27 GHz, 48G memory and 15k rpm SATA disks. The network connectivity is standard gigabit ethernet and all the servers are connected to the same network switch.
- For the server infrastructure we have measured configurations of 1, 2, 4, 8, 16, and 32 segments. For the cluster infrastructure we measured 4, 8, 16 and 32 - it makes no sense to measure fewer than 4 segments in a cluster made up of four physical nodes.

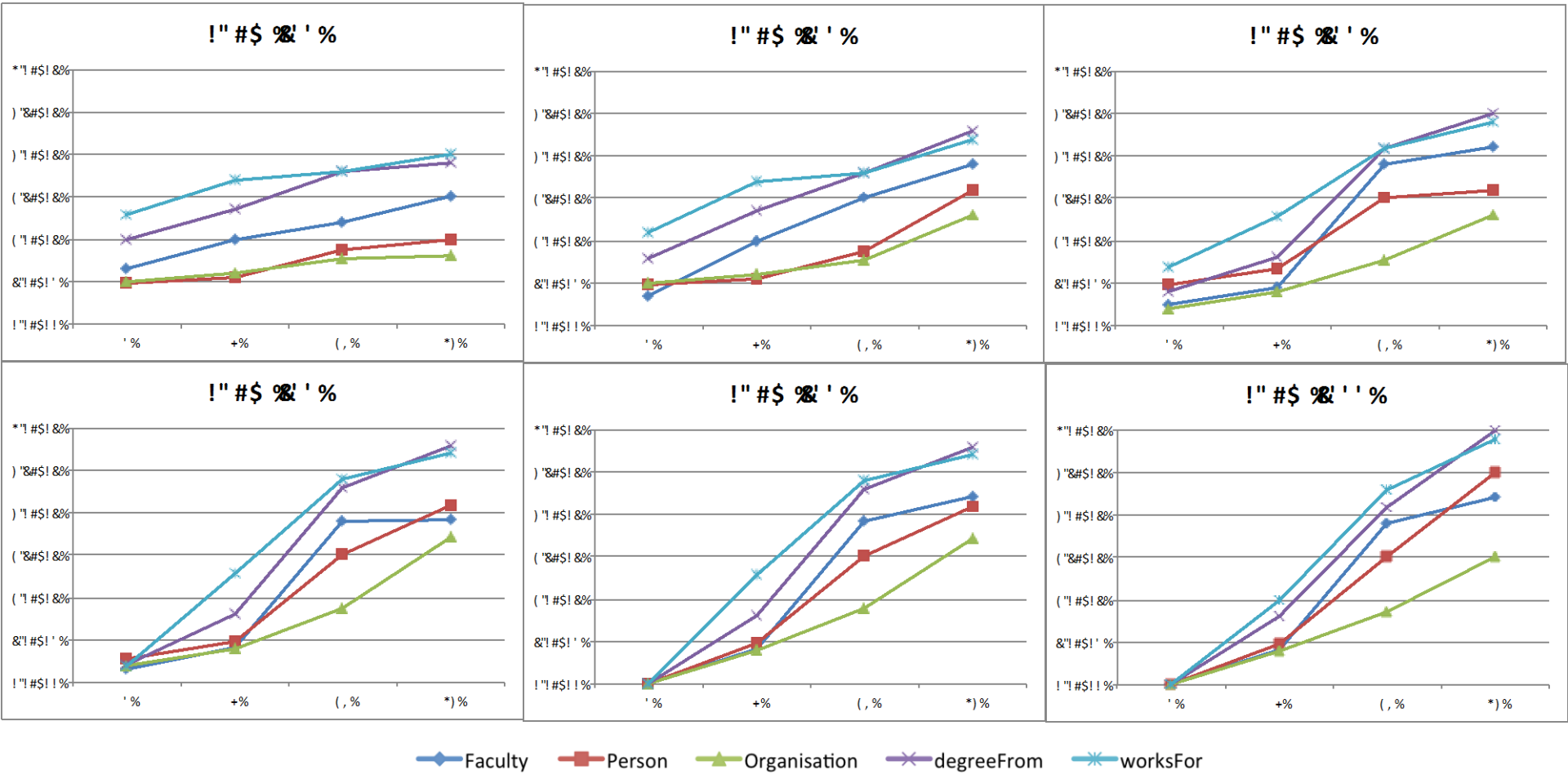
LUBM Scalability Evaluation

- **Faculty** {?s type Faculty}
- **Person** {?s type Person}
- **Organisation** {?s type Organisation}
- **degreeFrom** {?s degreeFrom ?o}
- **worksFor** {?s worksFor ?o}

LUBM Scalability Evaluation – server setup



LUBM Scalability Evaluation – cluster setup



Conclusions

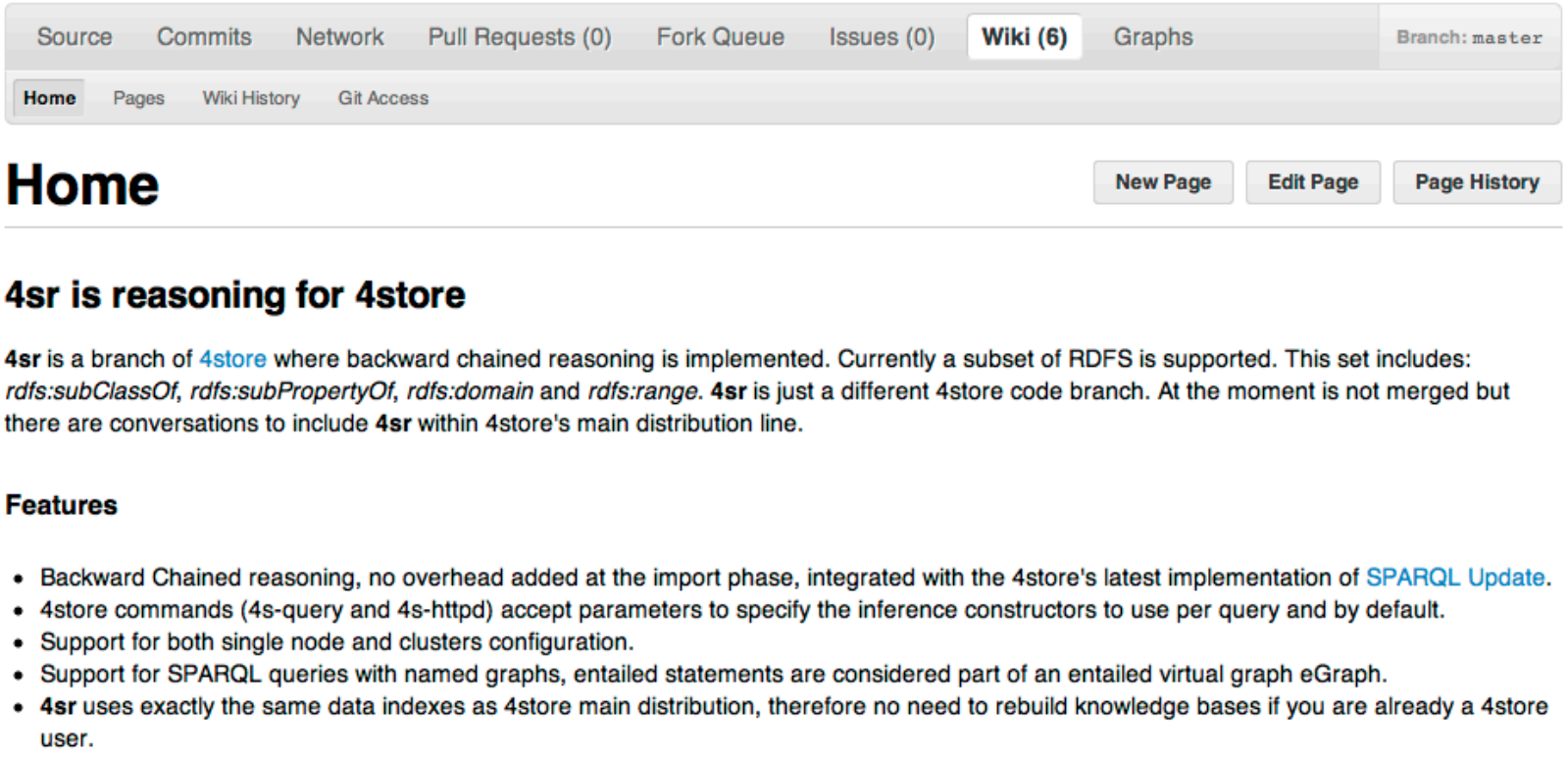
- Backward chained reasoning can scale in a distributed environment for Minimal RDFS and the ρdf fragment.
- 4sr can concurrently perform search in indexes (radix tries) with awareness of RDFS semantics by replicating a small subset of triples.
- The small subset of triples to replicate are the ones that use the ρdf constructors.
- Backward chain reasoning benefits:
  - More economic in space – number of quads.
  - No need to re-compute entailments between updates.

4sr latest release

http://4sreasoner.ecs.soton.ac.uk/

https://github.com/msalvadores/4sr/tree/rdfs-reasoner

https://github.com/msalvadores/4sr/wiki



Future Work

- Implement more OWL constructors by studying subsets to replicate sameAs, TransitiveProperty, inverseProperty, ...
- Merge with 4store main distribution. Probably with a compile option that will include RDFS reasoning.
- Look at overhead of subset replication when running SPARQL update(s).

Acknowledgments

- EnAKTing project www.enakting.org

- This work was supported by the EnAKTing project funded by the Engineering and Physical Sciences Research Council under contract EP/G008493/1.

Thank you,

Questions