

# Querying Datasets on the Web with High Availability

mtn2411 - Achilleas Oikonomopoulos  
mtn2406 - Ioannis Koutsoukis  
mtn2417 - Christina Tzortzaki



# About the Paper

- **Title:** Querying Datasets on the Web with High Availability
- **Authors:**
  - **Ruben Verborgh**, Ghent University – iMinds, Belgium
  - **Olaf Hartig**, University of Waterloo, Canada
  - **Ben De Meester**, Ghent University – iMinds, Belgium
  - **Gerald Haesendonck**, Ghent University – iMinds, Belgium
  - **Laurens De Vocht**, Ghent University – iMinds, Belgium
  - **Miel Vander Sande**, Ghent University – iMinds, Belgium
  - **Richard Cyganiak**, Digital Enterprise Research Institute, nui Galway, Ireland
  - **Pieter Colpaert**, Ghent University – iMinds, Belgium
  - **Erik Mannens**, Ghent University – iMinds, Belgium
  - **Rik Van de Walle**, Ghent University – iMinds, Belgium
- **Release Year:** 2014



# Motivation & Problem Statement

- The Web of Data is growing, but querying public datasets is unreliable.
- SPARQL endpoints have performance issues and low availability.
  - Servers become overloaded under high traffic.
- Need for a scalable and high-availability querying method.
- SPARQL endpoints struggle with availability (down ~1.5 days/month).
- Downloading full datasets is inefficient.

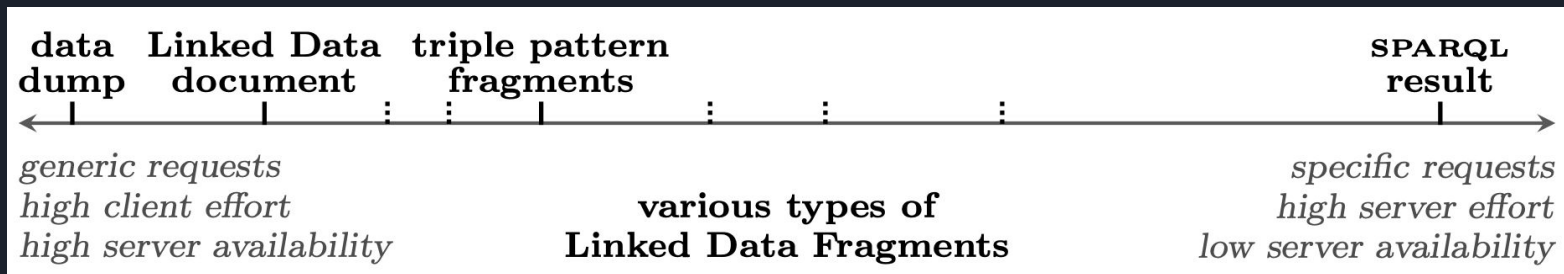


# Existing Querying Methods (HTTP Interfaces)

- SPARQL Endpoints: Flexible but unreliable.
- Linked Data: Simple but inefficient.
- RDF Dumps & APIs: Limited querying capabilities.
  - Need to download data locally

# Introduction to Linked Data Fragments (LDFs)

- LDFs are different ways of publishing & accessing data
  - SPARQL endpoints,
  - Linked Data documents
  - TPFs
  - + more
- Trade-off between server cost and query flexibility.





# Triple Pattern Fragments (TPFs)

- Special type of LDFs.
- Provide triple pattern-based access (subject, predicate, object).
- Include metadata and hypermedia controls.
  - Enable automatic navigation
  - Improve scalability
  - Encourage caching



# How TPFs Work

- Clients process queries instead of servers.
- Servers only provide simple fragments (lower computational cost).
- Improves availability at the cost of query speed.

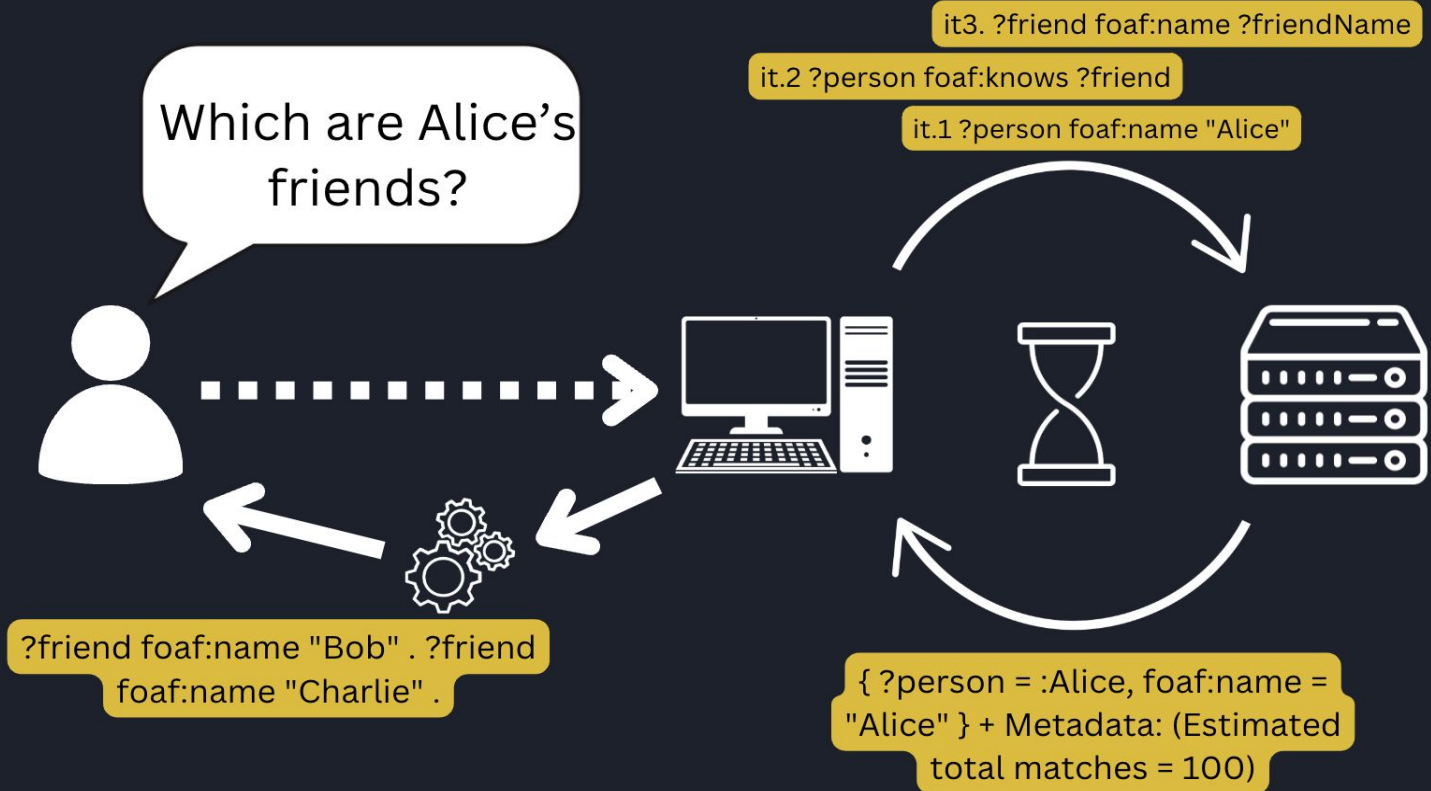


# Query Execution in TPFs

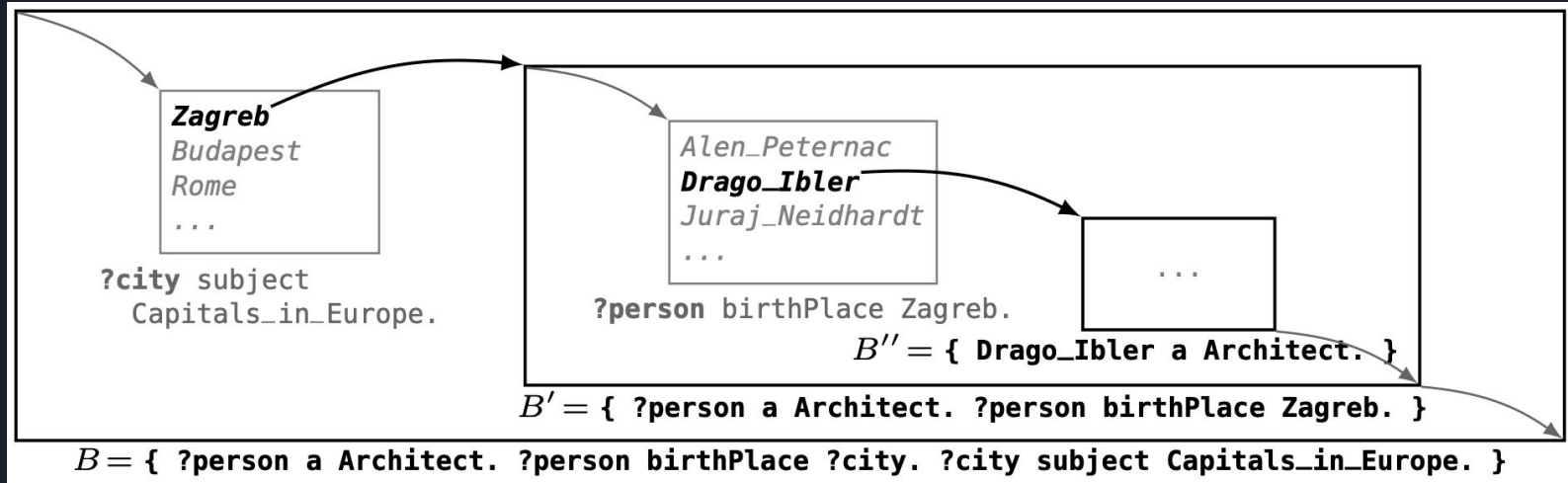
- SPARQL queries are transformed into sequences of triple pattern queries.
- Iterator-based query execution optimizes performance.
- Dynamic pipelines adjust query execution order dynamically.



# Dynamic Iterator Pipelines



# Example Case





# Evaluation & Results

Method	Query Power	Server Load	Availability	Speed	Client Load	Best For
SPARQL Endpoints	✓✓✓ Full SPARQL	✗ High	✗ Often fails	✗ Slow under load	✓ Low (server does all work)	Complex queries
Linked Data Documents	✗ Only single entities	✓ Low	✓ High	✗ Slow for large queries	✓ Low	Browsing RDF data
TPFs	✓ Limited to triple patterns	✓ Very low	✓✓ Very high	✗ Slower (many requests needed)	✗ High (client does query processing)	Scalable querying
<i>Data Dumps</i>	✗ No queries	✓ Zero	✓ Always available	✗ Slow (must process offline)	✗ High (must load entire dataset)	<i>Offline analysis</i>



# Advantages of the Algorithm

- + High availability (even under heavy traffic).
- + Lower server costs.
- + Compatible with HTTP caching.



# Limitations of the Algorithm

- Slower query execution.
- Client-side computation required.
- Limited support for complex queries (e.g., FILTER).



# Critical Discussion: Query Speed Trade-offs

- TPFs prioritize availability but at the cost of speed.
- Many real-time applications require both speed & availability.
- A hybrid approach with fallback SPARQL endpoints could work better.



# Critical Discussion: Limited Query Expressiveness


- TPFs only support exact matches, limiting:
  - FILTER expressions
  - Aggregation (COUNT, SUM)
  - ORDER BY, GROUP BY, LIMIT
- This leads to more HTTP requests, increasing latency.



# Critical Discussion: Client-Side Load

- Assumes clients can handle extra computation, but:
  - Low-End Clients, Mobile devices & IoT have limited resources.
  - Web applications need fast responses.





# Critical Discussion: Centralization Risks & Alternatives

- TPFs move work to clients, but who runs the servers?
- If only a few entities serve TPFs, we reintroduce centralization.
- Better Alternative: Federated TPF providers for decentralization.



# Conclusion & Future Directions

- TPFs shift query execution to clients, improving server availability.
- Ideal for large-scale Linked Data applications.
- Trade-off: Availability vs Query Speed.
- Future Work:
  - Supporting more SPARQL operators
  - Optimizing query planning for better performance
  - Hybrid approaches combining SPARQL & TPFs.



?you foaf:hasQuestion ?curiosity .

Thank you for your time!

