A Decentralized Architecture for Sharing and Querying Semantic Data

Christian Aebeloe   Gabriela Montoya   Katja Hose

Aalborg University, Denmark
{caebel,gmontoya,khose}@cs.aau.dk
Linked Data is often unavailable

Over half the public SPARQL endpoints have <95% availability\(^1\)

---

\(^1\)C. Buil-Aranda et al. SPARQL Web Querying Infrastructure: Ready for Action? ISWC 2013
INTRODUCTION

• Interfaces require *considerable resources* to maintain\(^2\)

---


\(^3\) T. Grubenmann et al. *Financing the Web of Data with Delayed-Answer Auctions.* WWW 2018
INTRODUCTION

- Interfaces require *considerable resources* to maintain\(^2\)
- *Huge burden* for data providers


\(^3\) T. Grubenmann et al. *Financing the Web of Data with Delayed-Answer Auctions*. WWW 2018
INTRODUCTION

• Interfaces require *considerable resources* to maintain\(^2\)
• *Huge burden* for data providers

• Recent efforts proposed *monetary incentives*\(^3\)

---


\(^3\) T. Grubenmann et al. *Financing the Web of Data with Delayed-Answer Auctions.* WWW 2018
INTRODUCTION

- Interfaces require *considerable resources* to maintain\(^{2}\)
- *Huge burden* for data providers

- Recent efforts proposed *monetary incentives*\(^{3}\)
- We argue, that the burden can be lifted through *decentralization*

---


\(^{3}\) T. Grubenmann et al. *Financing the Web of Data with Delayed-Answer Auctions*. WWW 2018
Decentralization has previously been proposed and evidenced to increase availability of data sources

- **Solid**\(^4\): Decentralized Personal Online Datastores (PODs)
  - Focuses on data privacy
- **TPF**\(^5\): Lower computational load on server
  - Single point of entry
- **Fog of Browsers**\(^6\): Sharing the load in a network of browsers
  - Browsers are relatively unstable, limited resources
- **Ulysses**\(^7\): Replicated TPFs over multiple servers
  - Fixed set of servers and replicated fragments

---


\(^6\) P. Molli et al. *Semantic Web in the Fog of Browsers*. DeSemWeb 2017

\(^7\) T. Minier et al. *Intelligent Clients for Replicated Triple Pattern Fragments*. ESWC 2018
PEER-TO-PEER SYSTEMS

• **Structured P2P Systems:**
  Organize peers into overlay networks (e.g. Distributed Hash Tables) to decide where to store and find data. Vulnerable to high churn.
PEER-TO-PEER SYSTEMS

- **Structured P2P Systems:**
  Organize peers into overlay networks (e.g. Distributed Hash Tables) to decide where to store and find data. Vulnerable to high churn.

- **Unstructured P2P Systems:**
  Unstructured connections between peers. More reliable during high churn. Allows for more dynamic behavior.
**PIQNIC**: a P2p client for Query processing over semantic data

- **Unstructured** P2P-based architecture

---

PIQNIC: a P2p client for Query processing over semantic data

- *Unstructured* P2P-based architecture
- Each client maintains a *local datastore*

---

8 J. D. Fernández et al. *Binary RDF Representation for Publication and Exchange (HDT).* J. Web Semantics 19:22-41 2013
PIQNIC: a P2p client for Query processing over semantic data

- Unstructured P2P-based architecture
- Each client maintains a local datastore
- HDT\textsuperscript{8} backend

PIQNIC: a P2p client for Query processing over semantic data

- **Unstructured** P2P-based architecture
- Each client maintains a *local datastore*
- **HDT** backend
- Queryable through *any node* in the network

---

Let $G_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment $f$ is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- $T$ is a finite set of RDF triples, and $T \subseteq G_N$,
- $N$ is a set of PIQNIC nodes containing the fragment,
- $u$ is a URI/IRI that identifies the fragment, and
- $i$ is an identification function that determines whether the fragment contains triples matching a given triple pattern.

\[
\begin{align*}
&\langle:Denmark :hasCapital :Copenhagen> \\
&\langle:Slovenia :hasCapital :Ljubljana> \\
&\langle:Greece :hasCapital :Athens> \\
&\langle:Germany :hasCapital :Berlin> \\
&\langle:Norway :hasCapital :Oslo>
\end{align*}
\]
Fragment
Let $G_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment $f$ is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- $T$ is a finite set of RDF triples, and $T \subseteq G_N$,
- $N$ is a set of PIQNIC nodes containing the fragment,
- $u$ is a URI/IRI that identifies the fragment, and
- $i$ is an identification function that determines whether the fragment contains triples matching a given triple pattern.

$<:\text{Denmark} :\text{hasCapital} :\text{Copenhagen}>$
$\quad <:\text{Slovenia} :\text{hasCapital} :\text{Ljubljana}>$
$\quad <:\text{Greece} :\text{hasCapital} :\text{Athens}>$
$\quad <:\text{Germany} :\text{hasCapital} :\text{Berlin}>$
$\quad <:\text{Norway} :\text{hasCapital} :\text{Oslo}>$

*Predicate-based* identification function
Let $G_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment $f$ is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- $T$ is a finite set of RDF triples, and $T \subseteq G_N$,
- $N$ is a set of PIQNIC nodes containing the fragment,
- $u$ is a URI/IRI that identifies the fragment, and
- $i$ is an identification function that determines whether the fragment contains triples matching a given triple pattern.

**Predicate-based** identification function

$$\langle ?v1:\text{hasCapital}:\text{Ljubljana} \rangle$$

**True**
Let $G_N$ be a knowledge graph that includes all RDF triples in a PIQNIC-network. A fragment $f$ is a 4-tuple $f = \langle T, N, u, i \rangle$ where

- $T$ is a finite set of RDF triples, and $T \subseteq G_N$,
- $N$ is a set of PIQNIC nodes containing the fragment,
- $u$ is a URI/IRI that identifies the fragment, and
- $i$ is an identification function that determines whether the fragment contains triples matching a given triple pattern.

**Predicate-based identification function**

$\langle ?v1 :\text{hasCapital} :\text{Ljubljana} \rangle$

$\langle ?v2 :\text{isLeaderOf} :\text{Slovenia} \rangle$

True

False
Dataset

A dataset $D$ is a triple $D = \langle F, u, o \rangle$ where

- $F$ is a set of fragments,
- $u$ is a URI/IRI that identifies the dataset, and
- $o$ is an identifier of the “owner” node, i.e., the node that uploads $F$ to the network.

\[
\begin{align*}
\langle &\text{Denmark} :\text{hasCapital} :\text{Copenhagen} > \\
&\langle &\text{Slovenia} :\text{hasCapital} :\text{Ljubljana} > \\
&\langle &\text{Greece} :\text{hasCapital} :\text{Athens} > \\
&\langle &\text{Germany} :\text{hasCapital} :\text{Berlin} > \\
&\langle &\text{Norway} :\text{hasCapital} :\text{Oslo} > \\
&\langle &\text{Oslo} :\text{locatedIn} :\text{Norway} > \\
&\langle &\text{Athens} :\text{locatedIn} :\text{Greece} > \\
&\langle &\text{Frank\_Jensen} :\text{isMayorOf} :\text{Copenhagen} > \\
&\langle &\text{Zoran\_Jankovic} :\text{isMayorOf} :\text{Ljubljana} > \\
&\langle &\text{Michael\_Muller} :\text{isMayorOf} :\text{Berlin} > \\
&\langle &\text{Ljubljana} :\text{hosted} :\text{ESWC} > \\
&\langle &\text{Athens} :\text{hosted} :\text{ESWC} > \\
&\langle &\text{Frank\_Jensen} :\text{isMarriedTo} :\text{Jane\_Pedersen} >
\end{align*}
\]
Dataset

A dataset $D$ is a triple $D = \langle F, u, o \rangle$ where

- $F$ is a set of fragments,
- $u$ is a URI/IRI that identifies the dataset, and
- $o$ is an identifier of the “owner” node, i.e., the node that uploads $F$ to the network.

<:Denmark :hasCapital :Copenhagen>
<:Slovenia :hasCapital :Ljubljana>
<:Greece :hasCapital :Athens>
<:Germany :hasCapital :Berlin>
<:Norway :hasCapital :Oslo>
<:Oslo :locatedIn :Norway>
<:Athens :locatedIn :Greece>

<:Frank_Jensen :isMayorOf :Copenhagen>
<:Zoran_Jankovic :isMayorOf :Ljubljana>
<:Michael_Muller :isMayorOf :Berlin>
<:Ljubljana :hosted :ESWC>
<:Athens :hosted :ESWC>
<:Frank_Jensen :isMarriedTo :Jane_Pedersen>
Given a knowledge graph, we would like to create a dataset

**Fragmentation Function**

A fragmentation function $\mathcal{F}$ is a function that, when applied to a knowledge graph $\mathcal{G}$, creates a set of fragments $F = \mathcal{F}(\mathcal{G})$, i.e.,

$$\mathcal{F}(\mathcal{G}) : \mathcal{G} \mapsto 2^\mathcal{G}.$$

**Predicate-Based Fragmentation Function**

$$\mathcal{F}_p(\mathcal{G}) = \{ F_p \mid \exists t \in \mathcal{G} : p_t = p \land (\forall t' \in \mathcal{G})[p_{t'} = p] : t' \in F_p \}$$

**Knowledge Graph $\mathcal{G}_E$**

```
<:Denmark :hasCapital :Copenhagen>
<:Slovenia :hasCapital :Ljubljana>
<:Greece :hasCapital :Athens>
<:Germany :hasCapital :Berlin>
<:Norway :hasCapital :Oslo>
<:Oslo :locatedIn :Norway>
<:Athens :locatedIn :Greece>
<:Frank_Jensen :isMayorOf :Copenhagen>
<:Zoran_Jankovic :isMayorOf :Ljubljana>
<:Michael_Muller :isMayorOf :Berlin>
<:Ljubljana :hosted :ESWC>
<:Athens :hosted :ESWC>
<:Frank_Jensen :isMarriedTo :Jane_Pedersen>
```
Given a knowledge graph, we would like to create a dataset

**Fragmentation Function**

A fragmentation function $\mathcal{F}$ is a function that, when applied to a knowledge graph $\mathcal{G}$, creates a set of fragments $\mathcal{F} = \mathcal{F}(\mathcal{G})$, i.e., $\mathcal{F}(\mathcal{G}) : \mathcal{G} \mapsto 2^{\mathcal{G}}$.

**Predicate-Based Fragmentation Function**

$\mathcal{F}_P(\mathcal{G}) = \{F_p | \exists t \in \mathcal{G} : p_t = p \land (\forall t' \in \mathcal{G})[p_{t'} = p] : t' \in F_p\}$

---

**Knowledge Graph $\mathcal{G}_E$**

- <:Denmark :hasCapital :Copenhagen>
- <:Slovenia :hasCapital :Ljubljana>
- <:Greece :hasCapital :Athens>
- <:Germany :hasCapital :Berlin>
- <:Norway :hasCapital :Oslo>
- <:Oslo :locatedIn :Norway>
- <:Athens :locatedIn :Greece>

**$f_1.T, f_2.T, f_3.T, f_4.T, f_5.T$**

- <:Frank_Jensen :isMayorOf :Copenhagen>
- <:Zoran_Jankovic :isMayorOf :Ljubljana>
- <:Michael_Muller :isMayorOf :Berlin>
- <:Ljubljana :hosted :ESWC>
- <:Athens :hosted :ESWC>

---

**A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA**
P2P network: clients are also servers

Node

A node $n$ is a triple $n = \langle \Gamma, \Delta, N \rangle$ where

- $\Gamma$ is the set of fragments located on the node,
- $\Delta$ is a set of datasets owned by the node, and
- $N$ is a set of so-called neighbor nodes in the network.
We want both *related* and *random* neighbors obtained through *shuffles*, i.e. we want to shuffle away the *least* related neighbors.

**Fragment Joinability**

Fragments $f_1$ and $f_2$ are *joinable*, denoted $f_1 \triangleleft f_2$, iff for at least one triple $t_1 \in f_1$ there exists a triple $t_2 \in f_2$, s.t. $\{s_{t_1}, o_{t_1}\} \cap \{s_{t_2}, o_{t_2}\} \neq \emptyset$.

$$Join(n_1, n_2) = \{f_1 \in n_1.\Gamma \mid \exists f_2 \in n_2.\Gamma : f_1 \triangleleft f_2 \wedge f_1.u \neq f_2.u\}$$
We want both related and random neighbors obtained through shuffles, i.e. we want to shuffle away the least related neighbors.

**Fragment Joinability**

Fragments $f_1$ and $f_2$ are joinable, denoted $f_1 \perp f_2$, iff for at least one triple $t_1 \in f_1$ there exists a triple $t_2 \in f_2$, s.t. $\{s_{t_1}, o_{t_1}\} \cap \{s_{t_2}, o_{t_2}\} \neq \emptyset$.

\[
\text{Join}(n_1, n_2) = \{f_1 \in n_1.\Gamma \mid \exists f_2 \in n_2.\Gamma : f_1 \perp f_2 \land f_1.u \neq f_2.u\}
\]

\[
\text{Rel}(n) = \arg \min_{R \subseteq n.N} \sum_{n_i \in n.N} \frac{|\text{Join}(n, n_i)|}{|n.\Gamma|} \quad \text{s.t. } |R| = k
\]

We select the $k$ least related neighbors to shuffle.
$r_i = |\text{Join}(n_4, n_i)| \div |n_4.\Gamma|$, $n_4$ shuffles with $n_2$, shuffle length = 1
NEIGHBORHOOD MANAGEMENT

A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA

\[ r_i = \frac{|\text{Join}(n_4, n_i)|}{|n_4 \cdot \Gamma|}, \text{ where } n_4 \text{ shuffles with } n_2, \text{ shuffle length } = 1 \]

- \( n_1: f_4 \perp f_1 \) and \( f_5 \perp f_2 \), then \( r_1 = \frac{2}{2} = 1 \)
A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA
A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA
NEIGHBORHOOD MANAGEMENT

\[ r_i = \frac{|\text{Join}(n_4, n_i)|}{|n_4.\Gamma|}, \text{ } n_4 \text{ shuffles with } n_2, \text{ shuffle length } = 1 \]

- \( n_1: f_4 \perp f_1 \) and \( f_5 \perp f_2 \), then \( r_1 = \frac{2}{2} = 1 \)
- \( n_2: f_4 \perp f_3 \) and \( f_5 \perp f_2 \), then \( r_2 = \frac{2}{2} = 1 \)
- \( n_3: f_4 \perp f_1, f_4 \perp f_3, f_5 \not\perp f_1 \) and \( f_5 \not\perp f_3 \), then \( r_3 = \frac{1}{2} = 0.5 \)
• Query processing follows a flooding approach:
  • i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)
• Query processing follows a flooding approach:
  • i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)
• Query processing follows a flooding approach:
  • i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)
• Query processing follows a flooding approach:
  • i.e. each triple pattern is *flooded* through the network for a specified amount of steps (Time-To-Live)
• We implemented three approaches:
  1. *Single*: Previous bindings are flooded one at a time
  2. *Bulk*: Previous bindings are flooded as bulks
  3. *Full*: Only the original triple patterns are flooded
QUERY PROCESSING - SINGLE

SELECT DISTINCT * WHERE {
    ?v1 :hasCapital ?v2 . (tp1)
    ?v2 :hosted :ESWC (tp2)
}

neighbors = 1, ttl = 3

A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}

neighbors = 1, ttl = 3

A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}

neighbors = 1, ttl = 3

\[f_1, f_2, f_3, f_4, f_5\]

\[\langle \text{Denmark} : \text{hasCapital} : \text{Copenhagen}\rangle, \langle \text{Norway} : \text{hasCapital} : \text{Oslo}\rangle, \langle \text{Slovenia} : \text{hasCapital} : \text{Ljubljana}\rangle, \langle \text{Germany} : \text{hasCapital} : \text{Berlin}\rangle, \langle \text{Greece} : \text{hasCapital} : \text{Athens}\rangle\]

\[\langle \text{Frank_Jensen} : \text{isMayorOf} : \text{Copenhagen}\rangle, \langle \text{Zoran_Jankovic} : \text{isMayorOf} : \text{Ljubljana}\rangle, \langle \text{Michael_Muller} : \text{isMayorOf} : \text{Berlin}\rangle\]

\[\langle \text{Ljubljana} : \text{hosted} : \text{ESWC}\rangle, \langle \text{Athens} : \text{hosted} : \text{ESWC}\rangle, \langle \text{Oslo} : \text{locatedIn} : \text{Norway}\rangle, \langle \text{Athens} : \text{locatedIn} : \text{Greece}\rangle\]

\[\langle \text{Frank_Jensen} : \text{isMarriedTo} : \text{Jane_Pedersen}\rangle\]
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}

eighbors = 1, ttl = 3

\begin{align*}
  & n_1 \quad f_1, f_2 \\
  & n_2 \quad f_2, f_3 \\
  & n_3 \quad f_1, f_3 \\
  & n_4 \quad f_4, f_5 \\
  & n_5 \quad f_2, f_4
\end{align*}

\textbf{tp}_2:
\{v_2 =: Ljubljana\},
\{v_2 =: Athens\}

\textbf{tp}_2 \bowtie \textbf{tp}_1:
\{v_2 =: Ljubljana, v_1 =: Slovenia\}
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)  
  ?v2 :hosted :ESWC   (tp2) 
}

neighbor = 1, ttl = 3

\( n_1 \) \( f_1, f_2 \) \( n_2 \) \( f_2, f_3 \)

\( n_3 \) \( f_1, f_3 \) \( n_4 \) \( f_4, f_5 \)

\( n_5 \) \( f_2, f_4 \)

\( \text{tp}_2: \)
\{\( v_2 =: \text{Ljubljana} \}, \{\( v_2 =: \text{Athens} \} \)

\( \text{tp}_2 \bowtie \text{tp}_1: \)
\{\( v_2 =: \text{Ljubljana}, v_1 =: \text{Slovenia} \} \)
\{\( v_2 =: \text{Athens}, v_1 =: \text{Greece} \} \)
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}

neighbors = 1, ttl = 3

```
\[ f_1, f_2, f_3, f_4, f_5 \]

\[ \langle \text{Denmark} : \text{hasCapital} : \text{Copenhagen} \rangle \]
\[ \langle \text{Norway} : \text{hasCapital} : \text{Oslo} \rangle \]
\[ \langle \text{Slovenia} : \text{hasCapital} : \text{Ljubljana} \rangle \]
\[ \langle \text{Germany} : \text{hasCapital} : \text{Berlin} \rangle \]
\[ \langle \text{Greece} : \text{hasCapital} : \text{Athens} \rangle \]
\[ \langle \text{Zoran_Jankovic} : \text{isMayorOf} : \text{Ljubljana} \rangle \]
\[ \langle \text{Frank_Jensen} : \text{isMayorOf} : \text{Copenhagen} \rangle \]
\[ \langle \text{Michael_Muller} : \text{isMayorOf} : \text{Berlin} \rangle \]
\[ \langle \text{Ljubljana} : \text{hosted} : \text{ESWC} \rangle \]
\[ \langle \text{Athens} : \text{hosted} : \text{ESWC} \rangle \]
\[ \langle \text{Oslo} : \text{locatedIn} : \text{Norway} \rangle \]
\[ \langle \text{Athens} : \text{locatedIn} : \text{Greece} \rangle \]
\[ \langle \text{Frank_Jensen} : \text{isMarriedTo} : \text{Jane_Pedersen} \rangle \]
```

\text{A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA}
**QUERY PROCESSING - BULK**

```sql
SELECT DISTINCT * WHERE {
?v1 :hasCapital ?v2 . (tp1)
?v2 :hosted :ESWC (tp2)
}
```

**neighbors = 1, ttl = 3**

```
tp2:
{v2 =: Ljubljana},
{v2 =: Athens}
```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}

neighbors = 1, ttl = 3

\[ n_1 \rightarrow n_2 \rightarrow n_3 \rightarrow n_4 \rightarrow n_5 \]

\[ f_1, f_2 \rightarrow f_2, f_3 \rightarrow f_1, f_3 \rightarrow f_4, f_5 \]

\[ tp_2: \{v_2 =: \text{Ljubljana}\}, \{v_2 =: \text{Athens}\}\]

\[ tp_2 \bowtie tp_1: \]
\[ \{v_2 =: \text{Ljubljana}, v_1 =: \text{Slovenia}\}\]
\[ \{v_2 =: \text{Athens}, v_1 =: \text{Greece}\}\]
SELECT DISTINCT * WHERE {
    ?v1 : hasCapital ?v2 . (tp1)
    ?v2 : hosted : ESWC (tp2)
}
A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA

```
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC       (tp2)
}
```

```
neighbors = 1, ttl = 3

n1

\[ f_1, f_2 \]

\[ n_2 \]

\[ f_2, f_3 \]

\[ n_3 \]

\[ f_1, f_3 \]

\[ n_4 \]

\[ f_4, f_5 \]

\[ n_5 \]

\[ f_2, f_4 \]

\[ n_5 \]

\[ f_2, f_4 \]

\[ \langle ?v_2, :hosted, :ESWC \rangle \]

\[ tp_2: \]

\[ \{ v_2 =: Ljubljana \}, \{ v_2 =: Athens \} \]
SELECT DISTINCT * WHERE {
  ?v1 :hasCapital ?v2 . (tp1)
  ?v2 :hosted :ESWC (tp2)
}

neighbors = 1, ttl = 3

\[
\text{tp}_2: \\
\{ v_2 =: Ljubljana \}, \{ v_2 =: Athens \} \\
\text{tp}_1: \\
\{ v_2 =: Ljubljana, v_1 =: Slovenia \} \\
\{ v_2 =: Athens, v_1 =: Greece \}
QUERY PROCESSING - FULL

```
SELECT DISTINCT * WHERE {
    ?v1 :hasCapital ?v2 . (tp1)
    ?v2 :hosted :ESWC  (tp2)
}
```

\( \text{neighbors} = 1, \text{ttl} = 3 \)

\[ f_1, f_2, f_3, f_4, f_5 \]

\[ f_1, f_2 \]

\[ f_2, f_3 \]

\[ f_1, f_3 \]

\[ f_4, f_5 \]

\[ f_2, f_4 \]

\[ tp_2 \bowtie tp_1:\]

\( \{v_2 =: \text{Ljubljana}, v_1 =: \text{Slovenia}\} \)

\( \{v_2 =: \text{Athens}, v_1 =: \text{Greece}\} \)
EXPERIMENTAL SETUP

• 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  • 200 nodes running concurrently

A. Hasnain et al. Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation. SSWS 2018
EXPERIMENTAL SETUP

• 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  • 200 nodes running concurrently
• LargeRDFBench\(^\text{11}\): Benchmark suite for federated query engines

\(^{11}\)A. Hasnain et al. Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation. SSWS 2018
EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - 200 nodes running concurrently
- LargeRDFBench\textsuperscript{11}: *Benchmark* suite for federated query engines
- 13 datasets with a total of over 1B triples

\textsuperscript{11} A. Hasnain et al. *Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation*. SSWS 2018
EXPERIMENTAL SETUP

- 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  - 200 nodes running concurrently
- LargeRDFBench\textsuperscript{11}: Benchmark suite for federated query engines
- 13 datasets with a total of over 1B triples
- 40 queries in 5 different categories
  - Cross Domain (CD): 7 queries
  - Life Sciences (LS): 7 queries
  - Complex (C): 10 queries
  - Larga-Data (L): 8 queries
  - Complex and High Amounts of Data Sources (CH): 8 queries

\textsuperscript{11} A. Hasnain et al. Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation. SSWS 2018
EXPERIMENTAL SETUP

• 4xAMD Opteron 6376 16 cores, 2.3GHz each, 516GB RAM
  • 200 nodes running concurrently
• LargeRDFBench\(^{11}\): Benchmark suite for federated query engines
• 13 datasets with a total of over 1B triples
• 40 queries in 5 different categories
  • Cross Domain (CD): 7 queries
  • Life Sciences (LS): 7 queries
  • Complex (C): 10 queries
  • Larga-Data (L): 8 queries
  • Complex and High Amounts of Data Sources (CH): 8 queries
• \(\text{ttl} = 5, \text{replication} = 5\%, \text{neighbors} = 5\)

\(^{11}\)A. Hasnain et al. Extending LargeRDFBench for Multi-Source Data at Scale for SPARQL Endpoint Federation. SSWS 2018
Bulk provides the best trade off between performance, traffic and data transfer.
Without recovery time, completeness decreases with the number of available nodes.
**ROBUSTNESS**

Without recovery time, completeness *decreases* with the number of available nodes.

![Graph showing decrease in completeness with decreased available nodes](image)

With recovery time, even when up to 50% of nodes were removed, we achieved at least 94.44% completeness.

![Graph showing maintained completeness with decreased available nodes](image)
ROBUSTNESS

*Without recovery* time, completeness *decreases* with the number of available nodes.

*With recovery* time, even when up to 50% of nodes were removed, we achieved at least 94.44% completeness.

**PIQNIC** is able to *retain* a high completeness during node failure.
CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

• *Piqnic*: a P2p client for Query processing over semantic data

Future Works

• Process queries with large intermediate results
• Assess the completeness of query answers during processing
• Alternative fragmentation and relatedness methods
CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- **Piqnic**: a P2p client for Query processing over semantic data
- Customizable approach to data fragmentation

Future Works

- Process queries with large intermediate results
- Assess the completeness of query answers during processing
- Alternative fragmentation and relatedness methods
CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

• *PiQnic*: a P2p client for Query processing over semantic data
• *Customizable* approach to data fragmentation
• Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*
Summary and Conclusion

- **PIQNIC**: a P2p client for Query processing over semantic data
- *Customizable* approach to data *fragmentation*
- Prototype implementation of three query processing approaches: *Single, Bulk* and *Full*
- *Bulk is better* overall and PIQNIC is able to *tolerate node failures*
CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

- **PIQNIC**: a P2p client for Query processing over semantic data
- Customizable approach to data fragmentation
- Prototype implementation of three query processing approaches: Single, Bulk and Full
- **Bulk is better** overall and PIQNIC is able to tolerate node failures

Future Works

- Process queries with **large intermediate results**
CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

• **PIQNIC**: a P2p client for Query processing over semantic data
• *Customizable* approach to data *fragmentation*
• Prototype implementation of three query processing approaches: *Single, Bulk and Full*
• *Bulk is better* overall and PIQNIC is able to *tolerate node failures*

Future Works

• Process queries with *large intermediate results*
• *Assess the completeness* of query answers during processing
CONCLUSIONS & FUTURE WORKS

Summary and Conclusion

• **PIQNIC**: a P2p client for Query processing over semantic data
• Customizable approach to data fragmentation
• Prototype implementation of three query processing approaches: *Single*, *Bulk* and *Full*
• *Bulk is better* overall and PIQNIC is able to tolerate node failures

Future Works

• Process queries with *large intermediate results*
• Assess the completeness of query answers during processing
• Alternative *fragmentation* and *relatedness* methods
Questions?
JOINING A NETWORK
ADDING A DATASET

A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA
ADDING A DATASET

A DECENTRALIZED ARCHITECTURE FOR SHARING AND QUERYING SEMANTIC DATA
FULL sends the fewest messages, SINGLE sends significantly more
Bulk transfers the least amount of data, Full transfers significantly more.
VARYING TIME-TO-LIVE VALUE

$t tl=5$ provides the best tradeoff between performance and completeness.
VARYING NUMBER OF NEIGHBORS

Number of Neighbors: 2 5 10

Execution Time (s)

CD1 CD2 CD3 CD4 CD5 CD6 CD7 LS1 LS2 LS3 LS4 LS5 LS6 LS7

No Results No Results No Results No Results No Results No Results No Results No Results No Results No Results No Results No Results No Results

$nn=5$ provides the best tradeoff between performance and completeness.
rep=5% provides the best tradeoff between performance and completeness