Web Semantics: Science, Services and Agents on the World Wide Web 16 (2012) 52–58

Contents lists available at SciVerse ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

SchemEX — Efficient construction of a data catalogue by stream-based indexing of linked data

Mathias Konrath, Thomas Gottron [*], Steffen Staab, Ansgar Scherp

WeST – Institute for Web Science and Technologies, University of Koblenz-Landau, D-56070 Koblenz, Germany[1]

ARTICLE INFO

Article history:
Received 30 January 2012
Received in revised form
11 May 2012
Accepted 12 June 2012
Available online 23 June 2012

Keywords:
Schema
Indexing
Stream processing
Linked Open Data
SchemEX
Billion Triples Challenge

ABSTRACT

We present SchemEX, an approach and tool for a stream-based indexing and schema extraction of Linked Open Data (LOD) at web-scale. The schema index provided by SchemEX can be used to locate distributed data sources in the LOD cloud. It serves typical LOD information needs such as finding sources that contain instances of one specific data type, of a given set of data types (so-called type clusters), or of instances in type clusters that are connected by one or more common properties (so-called equivalence classes). The entire process of extracting the schema from triples and constructing an index is designed to have linear runtime complexity. Thus, the schema index can be computed on-the-fly while the triples are crawled and provided as a stream by a linked data spider. To demonstrate the web-scalability of our approach, we have computed a SchemEX index over the Billion Triples Challenge (BTC) dataset 2011 consisting of 2,170 million triples. In addition, we have computed the SchemEX index on a dataset with 11 million triples. We use this smaller dataset for conducting a detailed qualitative analysis. We are capable of locating relevant data sources with recall between 71% and 98% and a precision between 74% and 100% at a window size of 100 K triples observed in the stream and depending on the complexity of the query, i.e. if one wants to find specific data types, type clusters or equivalence classes.

1. Introduction

Linked Open Data (LOD) [1] aims at publishing and connecting open data on the web using the Resource Description Framework (RDF). Data is provided by different, connected data sources using different publishing strategies like static RDF documents and SPARQL endpoints [1]. Altogether, LOD builds a huge web-scale RDF graph: the LOD cloud. This LOD cloud does not provide a single federated interface to perform graph queries. In conclusion this leads to the problem shown in Fig. 1(a) that for a given query it is not clear to which data sources this query should be sent in order to retrieve results. For instance, assume we wanted to find all people that are both politicians and actors and who are married with a model. While a query such as the one shown in Listing 1 can in principle provide the desired information, it is unclear which data source ds to address with such a query.

Furthermore, a complete retrieval of this huge RDF graph with currently more than 31 billion triples[2] to circumvent this problem and perform queries locally is not feasible either. Thus, one

```
SELECT ?x FROM ds WHERE {
  ?x rdf:type yago:AmericanFilmActors .
  ?x rdf:type dbpedia-owl:Politician .
  ?x dbpprop:spouse ?y .
  ?y rdf:type fbase:Model_(person) .
}
```

Listing 1: Example SPARQL query example on the LOD cloud

of the challenges when working with LOD is the lack of a concise summary or description of what kind of data can be found in which data source and how these data sources are connected. Such a summary is desirable for solving many tasks in common LOD scenarios, like searching, browsing, exploring, or querying the LOD graph. Typical tasks in these scenarios are, e.g., to find data sources that contain instances with certain properties, to detect which data sources are interlinked and to support the execution of distributed queries. To address this issue, some datasets provide a voiD description [2], containing metadata information such as a SPARQL endpoint location (`void:sparqlEndpoint`), example resources (`void:exampleResource`), and compositional relationships between different parts of the data source using the `dcterms:hasPart` and `dcterms:isPartOf` properties of Dublin Core.[3] However, even though voiD has found its way into

* Corresponding author. Tel.: +49 261 287 2862; fax: +49 261 287 100 2862.
E-mail addresses: mkonrath@uni-koblenz.de (M. Konrath), gottron@uni-koblenz.de (T. Gottron), staab@uni-koblenz.de (S. Staab), scherp@uni-koblenz.de (A. Scherp).

1 http://west.uni-koblenz.de

2 http://lod-cloud.net/

3 Dublin Core Metadata Initiative, http://dublincore.org/

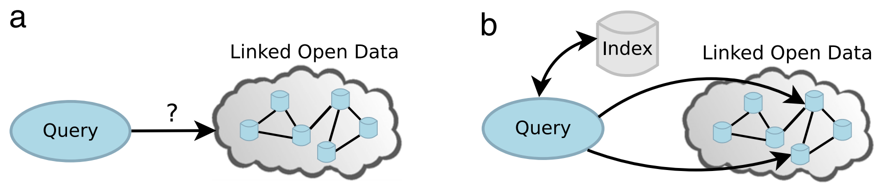

Fig. 1. Index usage for identifying relevant data sources.

the SPARQL 1.1 Service Description [3], to date only a fraction of datasets are publishing voiD descriptions, mostly triple store datasets such as DBPedia [4]. Neither does voiD contain explicit schema information. Thus, it is not sufficient to support all of the above tasks. A more precise index structure is desirable that allows us to search for data sources that contain instances with certain properties or to identify relevant data sources for a given query.

Our solution to this problem is to extract a concise schema from the LOD cloud with a suitable structure to be used as an index. In our context, schema extraction means to abstract RDF instances to RDF schema concepts that represent instances with the same properties. For using this schema as an index, each schema concept is mapped to data sources that contain instances with corresponding properties. This allows for searching for data sources that contain instances of a specific RDF type or instances that are connected via specific properties with other RDF types to find relevant data sources for given queries. Such a pre-processed schema-level index supports identifying the relevant data sources in the LOD cloud. It operates as service provider to find data sources as shown in Fig. 1(b).

In this paper, we introduce such an enhanced index structure called SchemEX that leverages RDF typings and links for creating a graph-based web-scale schema index. SchemEX uses a fixed-window approach for schema extraction and index building while operating on a stream of RDF triples. This allows for indexing without persistently storing the data and an effective integration with a Linked Data crawler such as LDSpider [5]. A crawl of 11 million triples which has been executed with LDSpider has been used for a detailed qualitative evaluation regarding precision and recall in locating relevant data sources when using our index. Additionally, the full Billion Triple Challenge (BTC) 2011 dataset with 2,170 million triples has been processed to demonstrate the scalability of SchemEX and the capability to handle web-scale data in web quality, i.e. to handle a very large datasets of varying quality and different origins. To the best of our knowledge such a stream-based schema extraction is novel in the LOD context.

2. Related work

Different approaches for indexing Linked Open Data have been developed in the past. Most of them can be divided into instance-level indices and schema-level indices. Instance-level indices contain knowledge about individual resources. This can be used to process and support queries which contain graph patterns with specific resources, e.g., querying all friends of Tim Berners-Lee. An example of an instance-level index is the QTree structure [6] that identifies relevant data sources for a given query that incorporates instance-level information. This is done by adding triples to corresponding buckets in the QTree. Queries are answered by finding relevant buckets for each query triple pattern and detecting overlaps between them. As SchemEX is a schema-level index, we concentrate in the following on schema-level indices.

2.1. Schema extraction and schema-level indices

Because instance-level indices contain knowledge about each individual instance, the size of the index grows with the size of the dataset. On the other hand, schema-level indices describe the structure of a dataset. Depending on the heterogeneity of the entities within a dataset, the corresponding schema is concise and reduced in size to a small fraction of the dataset. Because a schema does not contain instance-level knowledge, it cannot be used for answering such queries. But schemata can support queries by matching the query structure with sub-graph structures in the indexed dataset. Aggregated with lightweight ontologies like RDF-Schema, a schema can support the processing of queries like the Politician $\wedge$ Actor query given in Listing 1. Such a schema can be provided as a lookup service comparable to yellow pages.

Several approaches and algorithms have been developed for extracting schema information from graph data. Most approaches determine which data entities have an equivalent structure and assign these to a corresponding node in the schema graph. Links between the schema nodes are established if there is a link between at least one of their entities in the source graph. The approaches differ in the methodology and technical implementation of how the structural equivalence of entities is defined.

One example of such a schema is *DataGuide*, a concise and accurate summary describing the structure of a labelled and directed graph with a selected root node [7,8]. Entities are considered equivalent, if they can be reached from the root node via the same label paths. For each label path in the source graph a corresponding path in the schema graph is created. Every unique label path is described once in the DataGuide, regardless of the occurrence in the source graph. Computing a DataGuide is equivalent to conversion of a non-deterministic finite automaton into a deterministic finite automaton. In the worst case, it requires an exponential time complexity in dependency of the number of nodes and edges in the source graph.

An approach similar to DataGuide has been developed by Nestorov et al. [9]. Their algorithm uses greatest fixpoint semantics and is equivalent to a bottom-up clustering. Each entity is assigned to its individual and perfectly matching node in the schema. A distance function is used to count the structural differences between entities by looking at their outgoing edges and the types of the connected entities. If the distance function returns 0, the two compared entities are equivalent. Values >0 state that entities are not equivalent, but can be considered similar at small distance values. Using this distance function, equivalent entities are merged to a common node in the schema. Relaxing the stringency when comparing nodes can be reached by setting up a distance function threshold >0. This leads to an approximative schema in such a way that similar and slightly differing entities are grouped to the same schema node. The outcome is a schema with a significantly reduced number of nodes in comparison to a perfectly matching schema.

Another approach is to use bisimulation for schema extraction of graphs or XML data [10]. Bisimulation defines an equivalence relation on graph nodes based on their neighbourhood. Two nodes are considered equivalent and thereby assigned to the same equivalence class, if their sets of outgoing edges possess equal edge labels (some bisimulation approaches consider incoming edges as well) and point to instances which are equivalent themselves. This recursive definition describes a complete bisimulation. In such a bisimulation, the complete graph is considered as the neighbourhood of a selected node. Milo and Suciu base their 1-index [11] on a complete bisimulation. In contrast, the n-bisimulation only considers the neighbourhood within a restricted path length of $< n$. For example, the n-bisimulation is used by the $a(k)$-and $D(k)$-indices [12,13].

2.2. Schema extraction on LOD

In the context of Linked Open Data and Semantic Web, different approaches for schema extraction and building indices have been developed.

A voiD description can be considered as a schema — depending on the level of detail of the description — and used for index purposes. An approach for creating voiD descriptions [14] from web-scale datasets uses the Map-Reduce paradigm. In the first step, voiD datasets within the super-dataset are identified. Each voiD dataset represents RDF resources with corresponding properties. This could be, e.g. RDF resources of a certain RDF class or resources using the same set of properties or vocabularies. In a second Map-Reduce step links between the identified voiD datasets are determined. The created voiD descriptions can be used for supporting distributed queries.

ExpLOD is an approach for creating RDF usage summaries of RDF graphs [15]. The created summaries contain metadata about the structure of a RDF graph. This includes, e.g., the sets of instantiated RDF classes of a resource or the sets of used properties. This structure information is aggregated with statistics like the number of instances per class or the number of property usage. The ExpLOD summaries are extracted by partition refinement algorithms or alternatively via SPARQL queries. This procedure implies to have random access to the full dataset in order to compute the index.

The n-bisimulation (see Section 2.2) was applied on Linked Open Data by Tran et al. [16]. They used a parametrised L_1-forward–L_2-backward n-bisimulation, where L_1 and L_2 are subsets of the set of edge labels L. Only subsets of incoming and outgoing edges with certain labels are taken into account. The equivalence classes induced by this relation provide the elements of the index structure which are mapped to individual instances. In this setting, bisimulation does not regard RDF typings that usually exist and are available in LOD. The evaluation has shown that the 1-bisimulation provides the best trade-off between index size and response time [17]. The algorithm used takes a time complexity of $O(|L_1 \cup L_2||E| \log(|V|))$, where E and V denote the sets of edges and vertices.

3. SchemEX index

SchemEX defines a schema index structure consisting of three layers that contain different schema concepts. Each layer supports and enables different types of queries. In Sections 3.1–3.3, we introduce the index structure of SchemEX and define it formally. In Section 3.4, we compare the SchemEX index with the related work.

An example of the enhanced index structure underlying SchemEX is shown in Fig. 2. The index consists of three schema layers that reference the data source layer. The entries in the schema layer capture different types of schema information targeted at different types of queries. The schema information encoded into the entries is defined by class and link patterns observed in the RDF instances. The index then maps the elements from the layers to the data sources that actually provide the RDF data matching these patterns.

The source graph, i.e., the LOD cloud, is defined as a set of triples $T \subseteq V_{RB} \times P \times (V_{RB} \cup L)$, where $V_{RB} = (R \cup B)$ denotes the set of resources R and blank nodes B, P the set of RDF properties and L the set of literals. The source function $s : T \to \mathcal{P}(D)$ defines the mapping of a triple $t \in T$ to the set of data sources D that include it.

3.1. RDF class layer

The first layer of RDF classes captures schema information defined by the set $V_C = \{C \mid \langle i, \text{rdf:type}, C\rangle \in T \wedge i \in V_{RB}\} \subseteq R$ of individual *RDF classes* (e.g. `foaf:Person`) found in the dataset, indicated by C_1, C_2, C_3, and C_k in Fig. 2. This layer allows for supporting queries that select instances of a specified, single RDF class. Formally, we can map the elements C_j in this layer to data sources containing instances of those classes. To do so, we first define the set $SC_j = \{i \in V_{RB} \mid \exists \langle i, \texttt{rdf:type}, C_j\rangle \in T\}$ of all instances which belong to class C_j. Then we map C_j to all the data sources containing relevant triples having $i \in SC_j$ as subject by $C_j \mapsto \bigcup_{i \in SC_j} s(\langle i, p, o\rangle)$. This layer allows for retrieving all data sources that contain resources which are instances of a specific RDF class, e.g. instance of the class Politician or instance of the class Actor.

3.2. RDF type cluster layer

The second layer describes *type clusters*. Each type cluster TC is an element in the power set $\mathcal{P}(V_C)$ over the RDF classes in layer 1. To be more efficient, we use only those type clusters in the index that are actually observed in the data. The type clusters are mapped to those data sources providing instances that belong to an exact set of RDF classes such as TC_1, TC_2 and TC_m denoted in Fig. 2. A type cluster can be interpreted as a multi-inherited RDF class.

The type cluster of an instance i is defined by $\Gamma(i) := \{C \mid \langle i, \texttt{rdf:type}, C\rangle \in T\}$. Formally, the type clusters are mapped to data sources as defined by $TC \mapsto \bigcup_{i:\Gamma(i) \equiv TC} s(\langle i, p, o\rangle)$. For example, the type cluster layer provides the possibility to select data sources of RDF resources that are instances of the classes Politician and Actor at the same time.

3.3. Equivalence class layer

The third layer corresponds to *equivalence classes* as defined in bisimulation. Here, the type clusters are additionally partitioned into equivalence classes defined by outgoing RDF links and the type cluster of the target instances. Literals are derived and mapped to corresponding XML schema data types in the schema. These equivalence classes allow us to query over RDF properties with and without constraints on the RDF classes of subject and object. Formally, we define an equivalence relation $\sim$ on the instances. Two instances i_1 and i_2 are equivalent ($i_1 \sim i_2$), if and only if for every triple $\langle i_1, p, o_1\rangle$ a triple $\langle i_2, p, o_2\rangle$ exists (and vice versa), where $p \neq \texttt{rdf:type}$, $\Gamma(i_1) \equiv \Gamma(i_2)$ and $\Gamma(o_1) \equiv \Gamma(o_2)$.[4] The equivalence classes defined by $\sim$ define the structures in layer 3 and a particular equivalence class $[i]_\sim$ is then mapped to data sources by $[i]_\sim \mapsto \bigcup_{j \in [i]_\sim} s(\langle j, p, o\rangle)$.

Thus, as depicted in Fig. 2, RDF properties between instances in the data graph are represented in the schema as associations from the corresponding equivalence classes to type clusters such as property $p1$ that links EQC_1 and TC_2 and $p2$ that links EQC_2 and TC_m.

The equivalence class layer supports queries without `rdf:type` constraints only by the structure of the used properties. In the use of all three layers and the linkage between them, the index can, e.g., determine all data sources that contain data about people that are Politician and Actor at the same time and married with a Model.

3.4. Summary

While our index structure is based on equivalence classes induced by bisimulation, it extends previous approaches under

[4] It can easily be verified that this relation satisfies reflexivity, symmetry and transitivity.

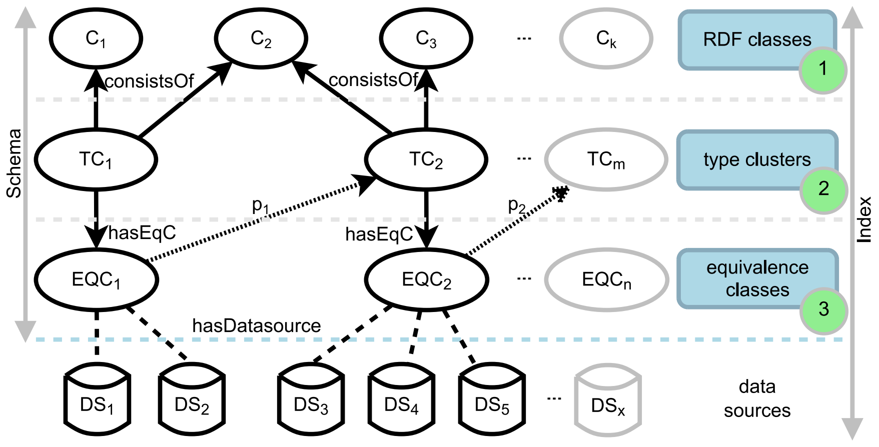

Fig. 2. Enhanced index structure with two additional layers leveraging RDF typings.

several aspects. On the one hand, our index structure adds two further schema layers to leverage RDF typings and incorporate lightweight ontologies. On the other hand, we use these additional layers to adapt the bisimulation itself to effectively incorporate RDF type information into the equivalence classes. Thereby, the index is extended to support a wider range of query types and improve and refine the results of type-selecting queries that use basic graph patterns with `rdf:type` predicate. Lightweight ontologies like RDF-Schema can be incorporated into the schema such as `rdf:subClassOf` and `rdf:subPropertyOf` relations. These can be used for inferencing and query refinement when querying the index.

4. SchemEX vocabulary

An extracted schema corresponding to the SchemEX structure can be represented in different output formats. Relational or graph databases, for instance, are a suitable framework for representing the index. In the context of LOD, it seems to be reasonable to use LOD standards for modelling and storing the schema as well. Thus, SchemEX uses RDF for representing and serialising the schema structure. This allows us to write the schema to a RDF file or directly to a triple store.

One of the principles of LOD is to reuse existing terms and vocabularies instead of inventing new ones [1]. Because of this, we use the voiD vocabulary for modelling the schema [2]. An example of a SchemEX structure modelled with voiD is shown in Fig. 3. Each type cluster is defined as a voiD dataset that represents all corresponding RDF resources. The `void:class` predicate is used to assign the RDF classes to a type cluster. The equivalence classes are modelled as voiD dataset and subset of a type cluster dataset. Relevant data sources are attached via `void:dataDump` links. voiD linksets are used to represent properties in this model. For each property a linkset node is created and the property itself attached via `void:property`, such as $p1$ in Fig. 3. `void:subjectsTarget` and `void:objectsTarget` are used to link equivalence classes and type clusters via the linkset.

Listing 2 shows the SPARQL query that can be applied on an index represented in the SchemEX vocabulary to determine relevant sources for the query in Listing 1. The query returns all data sources ds where one finds instances that are of type `yago:AmericanFilmActors` and `dbpedia-owl:Politician` and which have a `dbpprop:spouse` that is `fbase:Model_(person)`.

```
SELECT ?ds WHERE {
 ?tc void:class yago:AmericanFilmActors .
 ?tc void:class dbpedia-owl:Politician .
 ?tc void:subset ?eqc .
 ?eqc void:dataDump ?ds.
 ?link void:subjectsTarget ?eqc .
 ?link void:property dbpprop:spouse .
 ?link void:objectsTarget ?tc2 .
 ?tc2 void:class fbase:Model_(person) .
}
```

Listing 2: SchemEX example query

5. Stream-based computation of SchemEX index from LOD

In theory, the presented index can be built providing free access to the complete RDF graph that is indexed. This involves the need to lookup the RDF type information for each instance as well as for each of its referenced instances. Therefore, we have developed a scalable stream-based approach for schema extraction for the SchemEX index shown in Fig. 4. The stream-based processing allows for supplying the schema extractor directly with a quadruple stream (RDF triple + context/provenance URI) from a LOD crawler. To reach web-scale performance and a moderate memory consumption only a window of the stream with a certain width is observed and buffered by the schema extractor. This approach leverages the characteristic that a crawler traverses the graph from resource to resource via RDF properties so that linked instances are following in a certain distance within the stream. The buffer is implemented as a FIFO-cache (first in, first out) of RDF instances with a fixed size. Each incoming quadruple is parsed and assigned to the subject instance in the cache via its subject URI. If the instance does not exist in the cache yet, an instance object is created and appended to the FIFO-cache. The data structure used for the FIFO-cache is a ring queue and a hash map that is referenced by the ring queue. Adding a new and removing the oldest instance results in a time complexity of $O(1)$, because of the pointers on the first and last elements. All hash map operations — including random access via instance URI — result in a complexity of $O(1)$ as well. Due to this, all cache operations can be done in a constant runtime complexity of $O(1)$. If the cache is full, the oldest instance is removed, its schema information with respect to the three layers is derived and incorporated in the index.

Determining type clusters and equivalence classes is done by calculating an additional hash value. This is implemented by sorting the collected RDF classes and properties of each processed instance. The data structures used are red–black trees with time complexity of $O(c \cdot \log(c))$ ($c =$ number of RDF classes) respectively $O(p \cdot \log(p))$ ($p =$ number of properties). Hence, the hash calculation depends on the number of triples per instance and can be considered constant on average. Due to this stream-based approach, SchemEX has by design a linear runtime complexity with respect to the number of analysed triples.

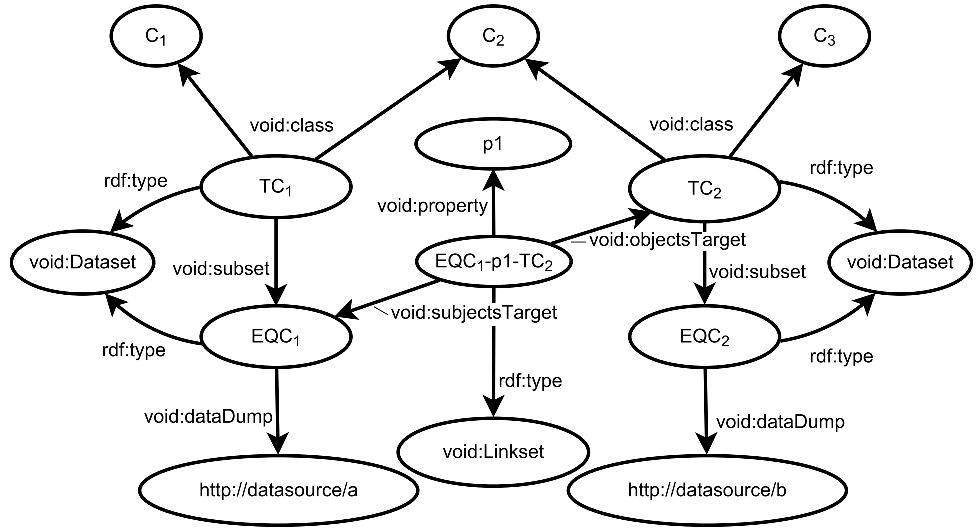

Fig. 3. SchemEX example modelled with voiD vocabulary.

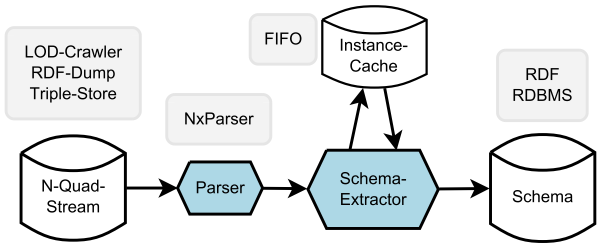

Fig. 4. SchemEX: architecture of stream-based processing.

The restriction to a certain window size of the data stream typically leads to incomplete results. This happens, if information about a single RDF instance is distributed over more triples in the stream sequence than the cache can hold. In this case, the instance has already been removed from the cache when we retrieve additional information potentially relevant for the schema. This occurs in particular when assigning an instance to an equivalence class as this assignment involves knowledge about the type clusters of all linked resources. Hence, the essential parameter for this approach is to choose an appropriate cache size value to obtain a certain degree of quality for the extracted schema.

6. Evaluation of SchemEX

We considered two datasets for different evaluation purposes. The TimBL dataset with 11 million triples is introduced and used in Section 6.1 to determine the completeness of the index constructed by stream-based means vs. a full index constructed with all data available in memory. The full BTC dataset[5] with 2.2 billion triples is used in Section 6.2 to show the applicability of the SchemEX approach on web-scale data. Both datasets have been generated using the same LOD crawler LDSpider [5].

6.1. *Qualitative evaluation of SchemEX on the TimBL dataset*

There are different query types that can be answered by the SchemEX index and that target the different schema layers introduced in Sections 3.1–3.3. These query types are of different complexity and are affected to a different degree by incomplete schema information as a consequence of the stream-based processing.

The simplest type of query retrieves all data sources providing instances of a certain type; therefore we refer to them as *RDF-Class* queries. Such queries can be answered by the RDF class layer (Section 3.1).

Slightly more information is needed to obtain those data sources providing instances which belong to a certain type cluster (*TC queries*, Section 3.2). An extension of these *TC* queries targets a set of RDF classes by considering all type clusters and super-type clusters that contain the RDF classes ($T + S - TC$, Sections 3.1 and 3.2). A super-type cluster is a type cluster composed of RDF classes of which a subset equates to another existing type cluster.[6]

The next type of query makes use of the third schema layer (Section 3.3) and addresses equivalence classes. Equivalence classes can be selected just by RDF properties (EQC) or in combination with type-selection constraints ($EQC + TC$). The latter query type is the most complex type and targets all schema layers.

To evaluate the index quality, we have constructed a gold standard on the TimBL dataset. This dataset was crawled with LDSpider starting at Tim Berner-Lee's FOAF file and aborted at a size of 11 million triples. Thus, the TimBL dataset has been created in the same fashion as the BTC data and thus can be assumed to have similar characteristics.

We described in Section 5 that our schema can be computed lossless for small datasets by using random access to the entire dataset and looking up the complete characteristics of all entities.[7] Such a lossless schema provides the gold standard to compare with the schema extracted by the SchemEX tool using the stream-based approach. The gold standard also provides all combinations of RDF classes, type clusters and equivalence classes present in the dataset. We compare all of these combinations in the gold standard with the schema extracted by SchemEX. Thereby, we can evaluate precision and recall over the relevant data sources for virtually all possible queries on the dataset.

The precision value for an individual query q is calculated by $Precision(q) = \frac{|D_{gold}(q) \cap D_{eval}(q)|}{|D_{eval}(q)|}$, where D_{gold} denotes the set of returned data sources of the reference schema and D_{eval} denotes the returned data sources of the schema extracted by SchemEX. Recall is calculated likewise by $Recall(q) = \frac{|D_{gold}(q) \cap D_{eval}(q)|}{|D_{gold}(q)|}$. The aggregated precision value for, e.g., all RDF classes is calculated by

5 Available from http://challenge.semanticweb.org

6 TCs with a subset of classes put less constraints on instances thus forming a larger set of instances.

7 For larger datasets, e.g. the full BTC dataset, it becomes infeasible to compute the gold standard.

Table 1
Schema extraction processes and results in dependency of window size (average of 10 runs).

	100	1 K	10 K	50 K	100 K	Gold/700 K
Runtime	182 s	223 s	192 s	194 s	203 s	376 s
Standard deviation	2.57 s	2.83 s	1.52 s	2.42 s	2.01 s	4.39 s
#Triples/s	60.4 K	49.3 K	57.3 K	56.7 K	54.2 K	29.3 K
Max. memory consumption	84 MB	136 MB	315 MB	731 MB	874 MB	3393 MB
#Type cluster	2772	2751	2749	2757	2761	2763
#Equivalence classes	13,570	12,885	12,281	12,062	12,184	11,955
#Triples voiD representation of index	270,871	241,187	246,396	255,751	263,916	277,695

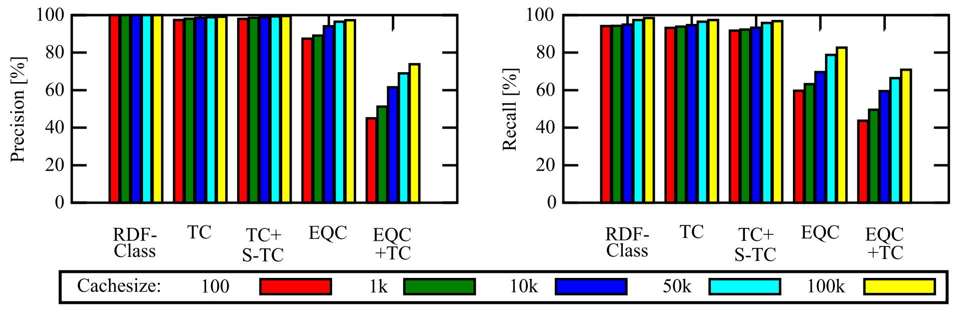

Fig. 5. Precision and recall for different query types in dependency on cache window size.

$Precision_{RDF\text{-}Classes} = \frac{1}{|V_C|}\sum_{\forall c \in V_C} Precision(\text{query-class}(c))$. Precision and recall values for other types of index queries are calculated analogously.

The lossy computation of SchemEX will affect both precision and recall, because a missing information fragment leads to a wrong type cluster or equivalence class classification. For example, if SchemEX misses the $\langle i, \text{rdf:type}, \mathsf{Politician}\rangle$ triple of a resource i that is instance of Politician and Actor, then i is classified wrongly to the type cluster *Actor*. If the triple $\langle i, \text{rdf:type}, \mathsf{Actor}\rangle$ follows later on, it is assigned to the type cluster Politician as well. The type clusters Actor and Politician have erroneous entries, while Actor $\wedge$ Politician misses a resource.

The gold standard on the TimBL dataset provides a total of about 674 K instances in the data graph, which are mapped to 2763 type clusters and about 12 K equivalence classes. Type clusters are partitioned by an average of 4.33 and up to a maximum of 1071 equivalence classes of common types like `foaf:Person`.

This evaluation of the quality of SchemEX is performed with respect to both the different query types and variable cache sizes. We have run the SchemEX approach with window sizes from 100 up to 100,000 instances and compared the generated index to the gold standard. Table 1 shows the statistics and values of the schema extraction process and the generated schemata. For each cache size, we computed ten runs in randomised order. In addition to the SchemEX index the numbers of the gold standard schema are shown.

As expected, the window size does not have a strong influence on the runtime performance. Regarding the maximum memory consumption, an increased window size by a factor of 10 results only in a 2–3 times higher memory consumption. This can be explained by observing more triples within a larger window size but these triples belong to relatively fewer new RDF classes.

The second part of the table shows values regarding the extracted schemata. With a higher window size, the values tend to converge towards the gold standard. As this is merely a quantitative analysis, the numbers do not guarantee that the counted type clusters and equivalence classes are exactly the same. This question is covered by the qualitative analysis based on precision and recall.

Fig. 5 shows precision and recall regarding the five query types introduced above. For each query type the values for different window sizes are shown. For the simpler type selecting queries (RDF-Class, $TC$and $TC + S - TC$) both precision ($>$98%) and recall ($>$91%) show very good results at any window size. On the more complex EQC and $EQC + TC$ type queries precision and recall are lower. However, with an increasing window size precision and recall can also be boosted to more than 70% for a window size of 100 K instances. Based on this evaluation, we chose a window size of 50 K instances for processing the second dataset from the BTC.

Regarding the evaluation results, it has to be considered that even wrongly classified instances are mostly assigned to very similar equivalence classes. Those wrongly classified instances may be returned correctly by EQC and $EQC + TC$ queries as they do not put constraints on all properties of an equivalence class.

6.2. BTC dataset schema extraction

To show that SchemEX can handle web-scale data of realistic web quality, we have extracted a schema from the Billion Triples Challenge 2011 dataset. We have processed the BTC data in two parts as well as the entire dataset. For our experiments, we have applied a cache size of 50 K instances. Table 2 shows the statistics of the schema extraction process and the extracted schemata. Each chunk of 10 M triples has been processed with an average of about 250 s using a single Intel Xeon CPU core with 2.93 GHz and 4 GB of RAM. This allows us to process 1 billion triples in $<$7 h and the full dataset in approximately 15 h.

The processing time demonstrates that SchemEX is suitable for real-time processing. Using commodity hardware, we have managed to realise a throughput of about 40 K triples per second which was mainly limited by reading the BTC dataset from disk. Thus, SchemEX can easily handle the stream generated by a LOD crawler (we have observed LDSpider providing about 2 K triples per second). Further, our stream-based approach has demonstrated to be able to handle web-scale data and can in principle be extended to an arbitrary amount of data.

Having such a schema for a very large dataset crawled from the web allows us to provide a look-up service for linked data clients to find relevant data sources on the web of data.

7. Conclusion

We have presented an approximating and efficient index structure for supporting distributed queries on Linked Open Data

Table 2
Characteristics of the BTC 2011 dataset (cache size = 50 K instances).

	1st billion	2nd billion	Full dataset
#Triples	1 billion	1 billion	2.17 billion
#Instances	187.7 M	222.6 M	450.0 M
#Data sources	13.5 M	9.5 M	24.1 M
#Type clusters	208.5 K	248.5 K	448.6 K
#Equivalence classes	0.97 M	1.14 M	2.12 M
#Triples voiD representation of index	29.1 M	24.8 M	54.7 M
Compression ratio	2.91%	2.48%	2.52%
Runtime (hh:mm)	6:51	6:05	15:16
Average runtime per 10M chunk	247 s	219 s	252 s
Standard deviation	80 s	12 s	57 s
#Triples/s	40.5 K	45.6 K	39.5 K

(LOD). Our SchemEX approach for schema extraction provides a good trade-off between scalability and result quality. Its stream-based processing allows for easily integrating with a LOD crawler. Future work will target on evaluating and optimising crawling strategies for gaining even better results. This entails the evaluation of other cache strategies for optimisation purposes. In addition, we need to investigate and implement options for maintaining, updating, and incrementally building a SchemEX index. While from a conceptual point of view these extensions should not be difficult to realise, we have not yet evaluated their potential impact on the index quality. As the developed index structure cannot deal with instance level queries, extending it in this direction would be of interest. Finally, SchemEX can be integrated with a federated query processing system and direct subqueries to relevant distributed data sources.

The schema extracted from the full BTC data is available on the web at http://west.uni-koblenz.de/schemex/.

Acknowledgements

We thank Andreas Harth for providing the crawled TimBL dataset that was used for the evaluation of SchemEX. We also thank Sebastian Rudolph for the discussions on bisimulation. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 257859, ROBUST and grant agreement no. 287975, SocialSensor.

References

[1] T. Heath, C. Bizer, Linked data: evolving the web into a global data space, in: Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool, 2011.
[2] K. Alexander, R. Cyganiak, M. Hausenblas, J. Zhao, Describing linked datasets with the void vocabulary, http://www.w3.org/TR/void/ (last visited 30.08.2011).
[3] G.T. Williams, Sparql 1.1 service description, W3c working draft, World Wide Web Consortium, Jan 2012.
[4] C. Bizer, A. Jentzsch, R. Cyganiak, State of the lod cloud, http://www4.wiwiss.fu-berlin.de/lodcloud/state/ (last visited 30.12.2011).
[5] R. Isele, A. Harth, J. Umbrich, C. Bizer, LDspider: an open-source crawling framework for the Web of Linked Data, in: Poster, International Semantic Web Conference 2010, Shanghai, China, 2010.
[6] A. Harth, K. Hose, M. Karnstedt, A. Polleres, K.-U. Sattler, J. Umbrich, Data summaries for on-demand queries over linked data, in: WWW, ACM, 2010, pp. 411–420.
[7] R. Goldman, J. Widom, Dataguides: enabling query formulation and optimization in semistructured databases, in: VLDB, Morgan Kaufmann, 1997, pp. 436–445.
[8] R. Goldman, J. Widom, Approximate dataguides, in: Proceedings of the Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats, vol. 97, 1999, pp. 436–445.
[9] S. Nestorov, S. Abiteboul, R. Motwani, Extracting schema from semistructured data, in: L.M. Haas, A. Tiwary (Eds.), SIGMOD Conference, ACM Press, 1998, pp. 295–306.
[10] S. Abiteboul, P. Buneman, D. Suciu, Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufmann, 1999.
[11] T. Milo, D. Suciu, Index structures for path expressions, in: ICDT, 1999, pp. 277–295.
[12] R. Kaushik, P. Shenoy, P. Bohannon, E. Gudes, Exploiting local similarity for indexing paths in graph-structured data, in: ICDE, 2002, pp. 129–140.
[13] Q. Chen, A. Lim, K.W. Ong, $D(k)$-index: an adaptive structural summary for graph-structured data, in: Proceedings of the 2003 ACM SIGMOD International Conference on Management of data, SIGMOD '03, ACM, New York, NY, USA, 2003, pp. 134–144. http://doi.acm.org/10.1145/872757.872776.
[14] C. Böhm, J. Lorey, F. Naumann, Creating void descriptions for web-scale data, J. Web Sem. 9 (3) (2011) 339–345.
[15] S. Khatchadourian, M.P. Consens, Explod: summary-based exploration of interlinking and rdf usage in the linked open data cloud, in: ESWC (2), 2010, pp. 272–287.
[16] T. Tran, P. Haase, R. Studer, Semantic search—using graph-structured semantic models for supporting the search process, in: ICCS, Springer, 2009, pp. 48–65.
[17] D.T. Tran, Semantic web search—a process-oriented perspective on data retrieval on the semantic web, Ph.D.Thesis, KIT, Fakultät für Wirtschaftswissenschaften, Karlsruhe, 2010.