Querying Distributed RDF Graphs: The Effects of Partitioning

Anthony Potter, Boris Motik, and Ian Horrocks

Oxford University
first.last@cs.ox.ac.uk

Abstract. Web-scale RDF datasets are increasingly processed using distributed RDF data stores built on top of a cluster of shared-nothing servers. Such systems critically rely on their data partitioning scheme and query answering scheme, the goal of which is to facilitate correct and efficient query processing. Existing data partitioning schemes are commonly based on hashing or graph partitioning techniques. The latter techniques split a dataset in a way that minimises the number of connections between the resulting subsets, thus reducing the need for communication between servers; however, to facilitate efficient query answering, considerable duplication of data at the intersection between subsets is often needed. Building upon the known graph partitioning approaches, in this paper we present a novel data partitioning scheme that employs minimal duplication and keeps track of the connections between partition elements; moreover, we propose a query answering scheme that uses this additional information to correctly answer all queries. We show experimentally that, on certain well-known RDF benchmarks, our data partitioning scheme often allows more answers to be retrieved without distributed computation than the known schemes, and we show that our query answering scheme can efficiently answer many queries.

1 Introduction

While the flexibility of the RDF data model offers many advantages, efficient management of large RDF datasets remains an open research topic. RDF data management systems can be conceived as single-machine systems constructed using techniques originating from relational databases [12, 2, 1, 20, 4], but the size of some RDF datasets exceeds the capacity of such systems. As a possible solution, distributed architectures based on a cloud of shared-nothing servers have been developed [9, 11, 21]. Such systems are promising, but research is still at a relatively early stage and scalability remains an open and critical problem.

The two main challenges in developing a distributed RDF system are (i) how to split up the data across multiple servers (i.e., data partitioning), and (ii) how to answer queries in a distributed environment (i.e., distributed query answering). These two challenges are closely connected: knowledge about data partitioning is relevant for query answering, and knowledge of typical query structure can be used to inform the data partitioning scheme. Nevertheless, one can investigate independently from query answering the extent to which a specific data

partitioning scheme reduces the need for distributed processing; for example, one can identify the percentage of the query answers that can be computed locally—that is, by evaluating queries on each server independently.

Data partitioning via hashing is a well-known partitioning scheme from the database community, and it has been applied in several RDF systems [21, 15, 4, 8, 13, 7]. In its simplest form, it distributes RDF data in a cluster by applying a hash function to a component of an RDF triple. Triples are commonly hashed by their subject to guarantee that *star queries* (i.e., queries containing only subject–subject joins) can be evaluated locally. Hashing has often been implemented using the MapReduce framework [3, 5, 19]; although hashing is typically not discussed explicitly in such systems, it is implicit in the map phase of distributed join processing. This approach, however, does not take into account the graph structure of RDF data, so nodes that are close together in the graph may not be stored on the same server, resulting in considerable distributed processing for many queries. As an alternative, an approach based on graph partitioning has been developed [9]. The goal of graph partitioning is to divide the nodes of the graph into several subsets while minimising the number of links with endpoints in different subsets. Thus, by partitioning RDF data using graph partitioning, one increases the chance that highly interconnected nodes are placed on the same server, which in turn increases the likelihood that query answers can be computed locally. This scheme, however, does not guarantee that common queries (such as star queries) can be evaluated locally, and it may thus require a significant amount of distributed computation to guarantee completeness, even in cases where locally computed answers fully answer the query.

These approaches can be augmented by duplicating data on partition boundaries [9, 11]. For sufficiently small queries, data duplication ensures that each query answer can be computed locally, and it can be used to provide the local evaluation guarantee for star-shaped queries in the graph partitioning setting [9]. Data duplication, however, can incur considerable storage overhead, potentially increasing the number of servers required to store a given RDF graph.

Our Approach We present a novel RDF data partitioning scheme that aims to reduce the need for distributed computation on common queries, but with minimal duplication and storage overhead; moreover, we present a query answering scheme that can correctly answer conjunctive queries over the partitioned data. Our main idea is to keep track of places where each data subset makes connections to other data subsets, and to exploit this information during query answering in order to identify possible non-local answers. In this way we can enjoy the benefits of graph partitioning and reduce the need for distributed processing, with only a minimal overhead of data duplication.

In this paper we focus mainly on the effects of our data partitioning scheme, which we experimentally show to be very promising on the LUBM [6] and the SP2B [16] benchmarks. For all queries from these two benchmarks, our data partitioning scheme ensures that a higher proportion of query answers can be retrieved without any distributed processing than with subject-based hashing [8, 13, 7] or the semantic hash partitioning (SHAPE) [11] approach. We do not

explicitly compare our approach with the original graph partitioning approach [9] because the SHAPE approach has already been shown to be more effective.

We also experimentally evaluate our query answering scheme, which we show can effectively answer many queries from the LUBM and SP2B benchmarks with little or no distributed processing. On some queries, however, our scheme becomes impractical as it requires the computation of a large number of partial matches (see Section 4 for details)—a drawback we look to overcome in our future work. These results are preliminary in the absence of a complete distributed system that would allow us to measure query processing times; instead, we measure the amount of work (in a sense that we make precise in Section 4) involved in distributed query processing.

2 Preliminaries

2.1 RDF

Let R be a set of *resources*. An *RDF term* is either a variable or a resource from R; a term is *ground* if it is not a variable. An *RDF atom* A is an expression of the form $\langle s, p, o\rangle$ where s, p, and o are RDF terms; the *vocabulary* of A is defined as $\mathsf{voc}(A) = \{s, p, o\}$; $\mathsf{var}(A)$ is the set of all variables in $\mathsf{voc}(A)$; atom A is *ground* if $\mathsf{var}(A) = \emptyset$; a *triple* is a ground atom; and an *RDF graph* G is a set of triples. The *vocabulary* of G is defined as $\mathsf{voc}(G) = \bigcup_{A \in G} \mathsf{voc}(A)$. As in the SPARQL query language, all variables in this paper start with a question mark. A *variable assignment* μ (or just *assignment*) is a partial mapping of variables to resources. For r a resource, let $\mu(r) = r$; for $A = \langle s, p, o\rangle$ an RDF atom, let $\mu(A) = \langle \mu(s), \mu(p), \mu(o)\rangle$; and for S a set of atoms, let $\mu(S) = \bigcup_{A \in S} \mu(A)$. The *domain* $\mathsf{dom}(\mu)$ of μ is the set of variables that μ is defined on; and the *range* $\mathsf{rng}(\mu)$ of μ is $\mathsf{rng}(\mu) = \{\mu(x) \mid x \in \mathsf{dom}(\mu)\}$. An *RDF conjunctive query* Q is an expression of the form (1), where each A_i, $1 \leq i \leq m$, is an RDF atom.

$$Q = A_1 \wedge \ldots \wedge A_m \tag{1}$$

By a slight abuse of notation, we often identify Q with the set of its atoms. The vocabulary and the set of variables of Q are defined as follows.

$$\mathsf{voc}(Q) = \bigcup_{1 \leq i \leq m} \mathsf{voc}(A_i) \qquad \mathsf{var}(Q) = \bigcup_{1 \leq i \leq m} \mathsf{var}(A_i) \tag{2}$$

We sometimes write queries using the SPARQL syntax. An *answer* to a query Q over an RDF graph G is an assignment μ such that $\mathsf{dom}(\mu) = \mathsf{var}(Q)$ and $\mu(Q) \subseteq G$; and $\mathsf{ans}(Q, G)$ is the set of all answers to Q over G. Note that our definitions do not support variable projection.

2.2 RDF Data Partitioning and Distributed Query Answering

A *partition* of an RDF graph G is an n-tuple of RDF graphs $\mathbf{G} = (G_1, \ldots, G_n)$ such that $G \subseteq G_1 \cup \ldots \cup G_n$. Each G_i is called a *partition element*, and n is the

size of $\mathbf{G}$; an *n-partition* is a partition of size n. One might expect a partition to satisfy $G = G_1 \cup \ldots \cup G_n$, but our relaxed condition allows us to capture our ideas in Section 3. Furthermore, we allow triples to be duplicated across partition elements—that is, we do not require $G_i \cap G_j = \emptyset$ for $i \neq j$. A *partitioning scheme* is a process that, given an RDF graph G, produces a partition $\mathbf{G}$. Given an RDF conjunctive query Q, an answer $\mu \in \mathsf{ans}(Q, G)$ is *local for* Q *and* $\mathbf{G}$ if some $1 \leq i \leq n$ exists such that $\mu \in \mathsf{ans}(Q, G_i)$; otherwise, μ is *non-local for* Q *and* $\mathbf{G}$. When Q and $\mathbf{G}$ are clear, we simply call μ local or non-local. Since non-local answers span partition elements, they are more expensive to compute if each partition element is stored on a separate server; hence, the main aim of a partitioning scheme is to maximise the number of local answers to common queries. The *quality* of a partition $\mathbf{G}$ for a set of queries $\mathbf{Q}$ is defined as the ratio of local answers to all answers for all of the queries in $\mathbf{Q}$ on $\mathbf{G}$. We often use this term informally, in which case we consider partition quality with respect to an unspecified set of queries that can be considered typical.

Allowing duplication of triples in partition elements can improve partition quality: given a non-local answer μ to a query Q, answer μ becomes local if we add $\mu(Q)$ to some partition element. However, duplication also increases storage overhead, and in the limit it can result in each partition element containing a complete copy of G. Hence, another aim of a partitioning scheme is to achieve a suitable balance between triple duplication and partition quality.

A *distributed query answering scheme*, or just a *query answering scheme*, is a process that, given a partition $\mathbf{G}$ and a query Q, returns a set of assignments $\mathsf{ans}(Q, \mathbf{G})$ such that $\mathsf{ans}(Q, \mathbf{G}) = \mathsf{ans}(Q, G)$. If the shape of Q guarantees that all answers are local, one can evaluate Q against each element of $\mathbf{G}$ and take the union of all answers; otherwise, additional work is required to identify non-local answers or to detect that no such answers exist. Query answering schemes differ mainly in how they handle the latter case: answer pieces are spread across multiple partition elements and they must be retrieved and joined together. We now have two clear goals for a query answering scheme: the first goal is to ensure correctness—that is, that $\mathsf{ans}(Q, \mathbf{G}) = \mathsf{ans}(Q, G)$—and the second goal is to minimise the amount of work required to construct non-local answers.

2.3 Existing Solutions

Now we present a brief overview of partitioning schemes and query answering schemes known in the literature.

Hashing is the simplest and most common data partitioning scheme [14, 18, 21]. Typically, a hashing function maps triples of an RDF graph to m buckets, each of which corresponds to a partition element. The hashing function is often applied to the triple's subject or the predicate, with subject being the most popular choice: this guarantees that triples with the same subject are placed together, ensuring that all answers to star queries are local.

Graph-based approaches exploit the graph structure of RDF data. In particular, one can use min-cut graph partitioning software such as METIS [10], which takes as input a graph G and the partition size n, and outputs n disjoint sets

of nodes of G such that all sets are of similar sizes and the number of edges in G connecting nodes in distinct sets is minimised. The approach by [9] reduces RDF data partitioning to min-cut graph partitioning to ensure that highly connected RDF resources are placed together into the same partition element, thus increasing the likelihood of a query answer being local.

One can combine an arbitrary data partitioning scheme with *n-hop duplication* to increase the proportion of local answers. Given an RDF graph G and a subgraph $H \subseteq G$, the n-hop expansion H_n of H with respect to G is defined recursively as follows: $H_0 = H$ and, for each $1 \leq i \leq n$,

$$H_i = H_{i-1} \cup \{\langle s, p, o\rangle \mid \langle s, p, o\rangle \in G \text{ and } \{s, o\} \cap \mathsf{voc}(H_{i-1}) \neq \emptyset\}. \quad (3)$$

While n-hop partitioning can considerably improve partition quality [9], it can also incur a substantial storage overhead. For example, even just 2-hop duplication can incur a storage overhead ranging from 67% to 435% [11]. Various optimisations have been developed to reduce this overhead, such as using directed expansion and excluding high degree nodes from the expansion.

Most data partitioning schemes are paired with a specific query answering scheme. Due to lack of space, we cannot present all such approaches in detail. Many of them have been implemented using MapReduce [3]—a framework for handling and processing large amounts of data in parallel across a cluster; a recent survey of MapReduce solutions can be found in [5]. Moreover, the Trinity.RDF system [21] uses Trinity [17]—a distributed in-memory key-value store.

3 Partitioning RDF Data

3.1 Aims

We now present our novel data partitioning scheme. Similar to [9], we use min-cut graph partitioning, but we extend the approach by recording the outgoing links in each partition element so as to facilitate the reconstruction of non-local answers. More specifically, we introduce a *wildcard* resource $*$, and use it to represent all resources 'external' to a given partition element. Thus, we know in each partition element which resources are connected to resources in other partition elements; we exploit this feature in Section 4 in order to obtain a correct query answering scheme. This allows us to attain a high degree of partition quality, while at the same time answering queries correctly without n-hop duplication.

The quality of partitions critically depends on the structure of the data and the anticipated query workload. Although application specific, we found the following assumptions to be common to a large number of applications.

Assumption 1. Subject–subject joins are common.

Assumption 2. Queries often constrain variables to elements of classes—that is, they often contain atoms of the form $\langle ?x, \mathit{rdf{:}type}, \mathit{class}\rangle$.

Assumption 3. Joins involving resources representing classes are uncommon—that is, queries rarely contain atoms $\langle ?x_1, \mathit{rdf{:}type}, ?y\rangle \wedge \langle ?x_2, \mathit{rdf{:}type}, ?y\rangle$.

Assumption 4. Joins on resources that are literals are uncommon—that is, if a query contains atoms $\langle ?x_1, :R, ?y\rangle \wedge \langle ?x_2, :S, ?y\rangle$, it is unlikely that a query answer will map variable $?y$ to a literal.

Assumption 5. The number of schema triples in G is small, so all schema triples can be replicated in each partition element.

Although n-hop duplication can increase partition quality [9, 11], it is often associated with a considerable storage overhead, particularly with real (as opposed to synthetic) RDF graphs. With this in mind, we formulate the following aims for our partitioning scheme:

Aim 1. maximise the number of local answers to star queries,

Aim 2. achieve similar, or better, partition quality than schemes employing n-hop duplication, and

Aim 3. minimise duplication, particularly compared to n-hop duplication.

3.2 Our Data Partitioning Scheme

Given an RDF graph G, let $*$ be a distinguished *wildcard* resource such that $* \notin \mathsf{voc}(G)$. Now let $V \subseteq \mathsf{voc}(G)$ be a subset of the vocabulary of G. Given a resource r, let $[r]_V = r$ if $r \in V$ and $[r]_V = *$ otherwise. Moreover, given an RDF atom $A = \langle s, p, o\rangle$, let $[A]_V = \langle [s]_V, [p]_V, [o]_V\rangle$. Finally, given an RDF graph G, let $[G]_V$ be the RDF graph defined by $[G]_V = \{[A]_V \mid A \in G\}$. In the rest of this section we formalise our data partitioning scheme, and in Section 4 we show how to use the wildcard resource to answer queries.

Instead of partitioning triples directly, we partition the vocabulary of the graph and use the result to construct a partition of the triples. More precisely, to construct an n-partition of G we first partition $\mathsf{voc}(G)$ into n subsets $V_1, \ldots, V_n$, and then we use these to construct a partition $\mathbf{G} = ([G]_{V_1}, \ldots, [G]_{V_n})$ of G. To ensure that $\mathbf{G}$ is a valid partition, we must select $V_1, \ldots, V_n$ such that

$$G \subseteq [G]_{V_1} \cup \ldots \cup [G]_{V_n} \tag{4}$$

holds. To achieve this, we first partition the vocabulary $\mathsf{voc}(G)$ into n disjoint sets $V'_1, \ldots, V'_n$, and then we extend these sets so that condition (4) is satisfied. This extension allows resources to be duplicated in multiple partition elements, which in turn means triples can also be duplicated. Typically, the duplicated triples are those that are on, or near, the border between partition elements. Since resource $*$ is not contained in $\mathsf{voc}(G)$, we use the subset relation in condition (4), rather than a more intuitive equality relation. Furthermore, our approach ensures that, for each partition element $G_i = [G]_{V_i}$ and each triple $\langle s, p, o\rangle \in G$, if $\{s, p, o\} \subseteq \mathsf{voc}(G_i)$, then $\langle s, p, o\rangle \in G_i$ holds. This property is not satisfied in previously known partitioning schemes, but it increases partition quality. We formalise these ideas using the following steps.

Step 1. Compute the undirected graph G' by removing from G all schema triples and triples containing class and literal resources (i.e., all triples of the form $\langle s, \mathit{rdf{:}type}, o\rangle$ and $\langle s, p, \ell\rangle$ with ℓ a literal), and by treating each remaining triple $\langle s, p, o\rangle$ as an undirected edge connecting s and o.

Step 2. Partition the nodes of G' into n disjoint sets using min-cut graph partitioning (e.g., using METIS), and let $V'_1, \ldots, V'_n$ be the resulting vocabularies.
Step 3. Extend each V'_i to $V_i^\star = V'_i \cup \{r \mid r \text{ occurs in a schema triple in } G\}$.
Step 4. Extend each $V_i^\star$ to $V_i = V_i^\star \cup \{o \mid \langle s, p, o\rangle \in G \text{ and } s \in V_i^\star\}$.
Step 5. Calculate $[G]_{V_i}$ for each V_i, and set $\mathbf{G} = \{[G]_{V_1}, \ldots, [G]_{V_n}\}$.

In Step 1, we takes into account Assumption 5 that schema triples can be replicated in each partition element. Furthermore, in line with our Assumptions 3 and 4 on our query workload, we do not expect triples to participate in joins on classes and literals; thus, we remove such triples in Step 1 so that the min-cut graph partitioning algorithm does not attempt to place resources connected via class or literal resources into the same partition element.

In order to satisfy (4), we must ensure that, for each triple $A \in G$, some V_i exists such that $\mathsf{voc}(A) \in V_i$. Thus, we must reintroduce the triples from G that correspond to edges in G' that were 'cut' during min-cut partitioning, as well as triples removed in Step 1. Thus, in Step 3 we introduce all resources occurring in the schema (including all classes and properties) into all partition elements; note that this ensures an efficient evaluation of queries mentioned in Assumption 2. Finally, since we assume that subject–subject joins are common (cf. Assumption 1), in Step 4 we reintroduce the missing triples into the partition element that contains the triple subject.

Partition element $[G]_{V_i}$ is the *core owner* of a resource r if $r \in V'_i$. Note that, if $[G]_{V_i}$ is the core owner of a resource r, then $[G]_{V_i}$ contains all triples in which r occurs in the subject position. Hence, if Q is a star query in which variable $?x$ participates in subject–subject joins, then all answers in which $?x$ is mapped to r can be obtained by evaluating Q in $[G]_{V_i}$; in other words, all answers to star queries are local, which is in line with our Aim 1.

3.3 An Example

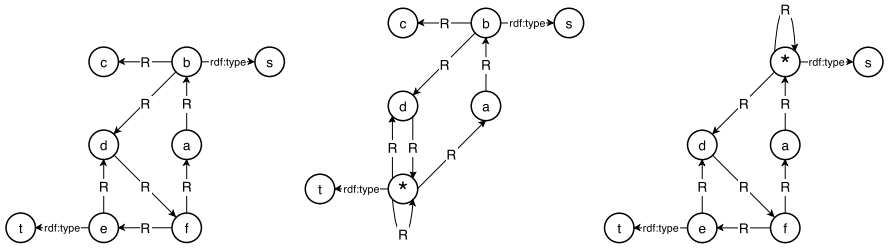
To make our scheme clear, we present an extended example. Let G be the RDF graph containing the following eight triples, shown schematically in Figure 1a.

$$\begin{aligned} G = \{\ &\langle a, R, b\rangle, \langle b, R, c\rangle, \langle b, R, d\rangle, \langle d, R, f\rangle, \langle e, R, d\rangle, \\ &\langle f, R, a\rangle, \langle f, R, e\rangle, \langle b, \mathit{rdf{:}type}, s\rangle, \langle e, \mathit{rdf{:}type}, t\rangle\} \end{aligned} \tag{5}$$

To produce a 2-partition of G, in Step 1 we first remove all triples containing class and literal resources; in our example, we remove triples $\langle b, \mathit{rdf{:}type}, s\rangle$ and $\langle d, \mathit{rdf{:}type}, t\rangle$. In Step 2 we then apply min-cut graph partitioning to the resulting graph to split the resources into two sets while minimising the number of cut edges; let us assume that this produces the following vocabularies:

$$V'_1 = \{a, b, c\} \qquad V'_2 = \{d, e, f\} \tag{6}$$

In Steps 3 and 4 we then extend these vocabularies so that each partition element that is a core owner of a subject also contains all triples with that subject, and



(a) Example Graph G (b) Partition Element $[G]_{V_1}$ (c) Partition Element $[G]_{V_2}$

Fig. 1: Example Graph G and the Resulting Partitioning Elements

so that each partition element includes all class and property resources; in our example, this produces the following vocabularies:

$$V_1 = \{a, b, c, d, s, t, R, \mathit{rdf{:}type}\} \qquad V_2 = \{a, d, e, f, s, t, R, \mathit{rdf{:}type}\} \tag{7}$$

Due to this step, nodes a, d, s and t as duplicated in V_1 and V_2; we explain using node d why this is necessary. Graph $[G]_{V_1}$ must contain all triples whose subject is in V_1'; thus, since $\langle b, R, d\rangle$ is in G and b is in V_1' , we must add d to V_1. Finally, we construct $[G]_{V_1}$ and $[G]_{V_2}$ as shown in Figures 1b and 1c.

4 Distributed Query Answering

Although there is considerable variation in the details, existing query answering schemes, such as [9, 11], generally proceed via the following steps: a query is broken up into pieces, all of which can be evaluated independently within partition elements; each query piece is evaluated in the relevant partition element to obtain partial matches; and the partial matches are then joined into query answers. As an example, consider the following query:

$$Q = \langle ?x, \mathit{rdf{:}type}, s\rangle \wedge \langle ?x, R, ?y\rangle \wedge \langle ?z, R, ?x\rangle \tag{8}$$

The data partitioning scheme critically governs the first step. For example, if the data partitioning scheme guarantees that subject–subject joins can be evaluated locally, then the query must be broken up into pieces each of which involves only subject–subject joins; thus, query Q will be broken into the following pieces:

$$Q_1 = \langle ?x, \mathit{rdf{:}type}, s\rangle \wedge \langle ?x, R, ?y\rangle \qquad Q_2 = \langle ?z, R, ?x\rangle \tag{9}$$

If the data partitioning scheme employs n-hop duplication, one can break the query into pieces that involve joins with n hops; however, as we show in Section 5, duplication can incur a considerable storage overhead.

The main drawback of such approaches is that they do not take advantage of local answers. For example, answer $\mu = \{?x \mapsto b, ?y \mapsto d, ?z \mapsto b\}$ is local with

respect to the partition $[G]_{V_1}$ shown in Figure 1b, but it would not be retrieved by evaluating Q on $[G]_{V_1}$ directly; instead, one must evaluate Q_1 and Q_2 on $[G]_{V_1}$ and then join the results. This can be problematical since evaluating query pieces might be less efficient than evaluating the entire query at once.

Our query answering scheme uses a completely different approach. Roughly speaking, we first evaluate each query in each partition element independently, thus retrieving all local answers without any partial query evaluation or communication between the servers. However, in this step we may also retrieve answers containing the wildcard resource, each of which represents a potential match of the query across partition elements, so we join such answers to obtain all answers to the query. In this way, we restrict communication and partial query evaluation to (possible) non-local answers, rather than all answers.

4.1 Formalisation

To formalise our query answering scheme, we first introduce some notation. Let V be a vocabulary not containing $*$. For Q a conjunctive query of the form (1), let $[Q]_V = [A_1]_V \wedge \ldots \wedge [A_m]_V$. Furthermore, for μ a variable assignment, let $[\mu]_V$ be the variable assignment such that $\mathsf{dom}([\mu]_V) = \mathsf{dom}(\mu)$ and, for each variable $x \in \mathsf{dom}(\mu)$, we have $[\mu]_V(x) = \mu(x)$ if $\mu(x) \in V$, and $[\mu]_V(x) = *$ if $\mu(x) \notin V$.

In the rest of this section, we fix an arbitrary conjunctive query Q of the form (1) with m atoms, an arbitrary RDF graph G, and an arbitrary partition $\mathbf{G} = ([G]_{V_1}, \ldots, [G]_{V_n})$ of G. To evaluate Q in $\mathbf{G}$, we first evaluate $[Q]_{V_i}$ in $[G]_{V_i}$ for each $1 \leq i \leq n$. Note that query Q may contain resources not contained in V_i; therefore, in each partition element $[G]_{V_i}$ we evaluate $[Q]_{V_i}$, rather than Q. We then join all answers obtained in the previous step, while assuming that resource $*$ matches any other resource. We formalise the join procedure as follows.

Definition 1. *A variable assignment μ is a* join *of assignments μ_1 and μ_2, written $\mu = \mu_1 \bowtie \mu_2$, if $\mathsf{dom}(\mu) = \mathsf{dom}(\mu_1) = \mathsf{dom}(\mu_2)$ and, for each $x \in \mathsf{dom}(\mu)$, (i) $\mu_1(x) = \mu_2(x)$ implies $\mu(x) = \mu_1(x) = \mu_2(x)$, and (ii) $\mu_1(x) \neq \mu_2(x)$ implies $\mu_1(x) = *$ and $\mu(x) = \mu_2(x)$, or $\mu_2(x) = *$ and $\mu(x) = \mu_1(x)$.*

An assignment can be an answer to Q on $\mathbf{G}$ only if it instantiates all atoms of Q, which we formalise as follows.

Definition 2. *Let μ be a variable assignment with $\mathsf{dom}(\mu) = \mathsf{var}(Q)$. The set of* valid atoms *of Q under μ is defined as $\mathsf{val}_\mu(Q) = \{j \mid * \notin \mathsf{voc}(\mu(A_j))\}$. Moreover, μ is* valid *for Q if $|\mathsf{val}_\mu(Q)| = m$.*

Furthermore, when evaluating $[Q]_{V_i}$ in $[G]_{V_i}$, we can ignore any variable assignment $\mu_1 \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$ that is redundant according to the following definition. Intuitively, μ_1 is redundant if, for each $\mu \in \mathsf{ans}(Q, G)$ that 'extends' μ_1 (i.e., such that $\mu_1 = [\mu]_{V_i}$), there exists a variable assignment μ_2 obtained by evaluating $[Q]_{V_j}$ in some partition element $[G]_{V_j}$ such that μ extends μ_2, and the set of atoms of Q fully instantiated by μ_2 strictly includes the set of atoms fully instantiated by μ_1. Note that this includes the case where no $\mu \in \mathsf{ans}(Q, G)$ extends μ_1. This idea is formally captured using the following definition.

Definition 3. *Consider arbitrary* $1 \leq i \leq n$ *and* $\mu_1 \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$*. Assignment* μ_1 *is* redundant *for* Q *and* i *if, for each assignment* $\mu \in \mathsf{ans}(Q, G)$ *such that* $\mu_1 = [\mu]_{V_i}$*, there exist* $1 \leq j \leq n$ *and an assignment* $\mu_2 \in \mathsf{ans}([Q]_{V_j}, [G]_{V_j})$ *such that* $\mu_2 = [\mu]_{V_j}$ *and* $\mathsf{val}_{\mu_1}([Q]_{V_i}) \subsetneq \mathsf{val}_{\mu_2}([Q]_{V_j})$*. If such* μ_1 *is neither valid for* Q *nor redundant for* Q *and* i*, then* μ_1 *is a* partial match of Q in $[G]_{V_i}$*.*

As a simple consequence of Definition 3, note that the number of non-redundant answers in each partition element is at most equal to the number of non-local query answers. Theorem 1 captures the essence of our query answering scheme. Intuitively, it says that each answer μ to Q on G is obtained either by evaluating $[Q]_{V_i}$ on some partition element $[G]_{V_i}$ (i.e., it is a local answer), or by joining non-valid and non-redundant assignments $\mu_1, \ldots, \mu_n$ obtained by evaluating $[Q]_{V_i}$ on $[G]_{V_i}$ that instantiate all atoms of Q.

Theorem 1. *For a variable assignment* μ*, we have* $\mu \in \mathsf{ans}(Q, G)$ *if and only if*

1. μ *is valid for* Q *and* $\mu \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$ *for some* $1 \leq i \leq n$*, or*
2. *variable assignments* $\mu_1, \ldots, \mu_n$ *exist such that*
   (a) *for each* $1 \leq i \leq n$*, either* $\mathsf{dom}(\mu_i) = \mathsf{var}(Q)$ *and* $\mathsf{rng}(\mu_i) = \{*\}$*, or we have* $\mu_i \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$ *and* μ_i *is a partial match of* Q *in* $[G]_{V_i}$*,*
   (b) *for each* $1 \leq j \leq m$*, some* $1 \leq k \leq n$ *exists such that* $j \in \mathsf{val}_{\mu_k}([Q]_{V_k})$*, and*
   (c) $\mu = \mu_1 \bowtie \ldots \bowtie \mu_n$*.*

Proof. ($\Rightarrow$) Assume that $\mu \in \mathsf{ans}(Q, G)$. The claim holds trivially if μ satisfies (1), so assume that μ does not satisfy (1). For each $1 \leq i \leq n$, let $\xi_i = [\mu]_{V_i}$, and let μ_i be such that $\mathsf{dom}(\mu_i) = \mathsf{var}(Q)$ and $\mathsf{rng}(\mu_i) = \{*\}$ if ξ_i is redundant for Q and $\mu_i = \xi_i$ otherwise. We next show that each μ_i satisfies (2a)–(2c).

(2a) Consider an arbitrary $1 \leq i \leq n$. The claim holds trivially if ξ_i is redundant for Q and i, so we assume that $\mu_i = \xi_i$ is not redundant for Q and i. Since we assume that $\mu \notin \mathsf{ans}(Q, G)$, assignment ξ_i is not valid for Q. For each $1 \leq j \leq m$, since $\mu(A_j) \in G$, we clearly have $[\mu(A_j)]_{V_i} \in [G]_{V_i}$; furthermore, we have $[\mu(A_j)]_{V_i} = \mu_i([A_j]_{V_i})$, so $\mu_i([A_j]_{V_i}) \in [G]_{V_i}$ holds. Consequently, we have $\mu_i \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$, as required.

(2b) Consider an arbitrary $1 \leq j \leq m$; then, $\mu \in \mathsf{ans}(Q, G)$ clearly implies $\mu(A_j) \in G$. Since $G \subseteq \bigcup_i [G]_{V_i}$, some $1 \leq i \leq n$ exists such that $\xi_i(A_j) \in [G]_{V_i}$, so clearly $j \in \mathsf{val}_{\xi_i}([Q]_{V_i})$. Now choose $1 \leq k \leq n$ such that $\mathsf{val}_{\mu_k}([Q]_{V_k})$ is a largest set satisfying $\mathsf{val}_{\xi_i}([Q]_{V_i}) \subseteq \mathsf{val}_{\xi_k}([Q]_{V_k})$. Since $\mathsf{val}_{\mu_k}([Q]_{V_k})$ is largest, such ξ_k is not redundant for Q and k, so $\mu_k = \xi_k$; but then, $j \in \mathsf{val}_{\mu_k}([Q]_{V_k})$, as required.

(2c) For each variable $x \in \mathsf{dom}(\mu)$, some $1 \leq i \leq n$ exists such that $\mu(x) \in V_i$; hence, it is obvious that $\mu = \xi_1 \bowtie \ldots \bowtie \xi_n$ holds. We next show that we can successively replace in this equation each ξ_i that is redundant for Q and i with μ_i. To this end, choose an arbitrary ξ_i that is redundant for Q and i, and choose an arbitrary $1 \leq j \leq n$ such that $\mathsf{val}_{\xi_i}([Q]_{V_i}) \subseteq \mathsf{val}_{\xi_j}([Q]_{V_j})$ and ξ_j is not redundant for Q and j; clearly, we have $[\xi_i]_{V_j} \bowtie \mu_j = \mu_j$. Now consider an arbitrary variable $x \in \mathsf{dom}(\xi_i)$ such that $\xi_i(x) \neq *$ and $\xi_i(x) \neq \xi_j(x)$. Since $\mathsf{val}_{\xi_i}([Q]_{V_i}) \subseteq \mathsf{val}_{\xi_j}([Q]_{V_j})$ holds, variable x occurs in Q only in atoms A_ℓ such

that $* \in \mathsf{voc}(\xi_i(A_\ell))$, so $\ell \notin \mathsf{val}_{\xi_i}([Q]_{V_i})$. But then, by (2b), some $1 \leq k \leq n$ exists such that $\xi_k(x) = \xi_i(x)$ and ξ_k is not redundant for Q and k. But then, $\mu = \xi_1 \bowtie \ldots \xi_{i-1} \bowtie \mu_i \bowtie \xi_{i+1} \bowtie \xi_n$ clearly holds. We can iteratively replace in this equation each ξ_i that is redundant for Q and i with μ_i without affecting the equality, as required.

($\Leftarrow$) Assume that (1) is true for some μ; then $\mu([A_j]_{V_i}) = \mu(A_j)$ for each $1 \leq j \leq m$, so clearly $\mu \in \mathsf{ans}(Q, G)$. Assume now that (2a)–(2c) are true, and consider an arbitrary $1 \leq j \leq m$. By (2a) and (2b), some $1 \leq i \leq n$ exists such that $\mu_i([A_j]_{V_i}) \in [G]_{V_i}$ and $* \notin \mathsf{voc}(\mu_i([A_j]_{V_i}))$. But then, $\mu_i([A_j]_{V_i}) \in G$; furthermore, by (2c), $\mu_i(x) = \mu(x)$ for each variable $x \in V_j$, so $\mu(A_j) \in G$. Consequently, $\mu \in \mathsf{ans}(Q, G)$. □

Hence, the answers to a query Q over $\mathbf{G}$ can be computed as follows. First, each i-th server computes $\mathsf{ans}([Q]_{V_i}, [G]_{V_i})$ in parallel, and it immediately returns all answers that are valid for Q. Second, the server identifies a subset $P_i \subseteq \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$ of partial matches of Q in $[G]_{V_i}$, and it also extends P_i with assignment μ such that $\mathsf{dom}(\mu) = \mathsf{var}(Q)$ and $\mathsf{rng}(\mu) = \{*\}$. Third, all servers communicate P_i to one designated server, which then computes the join $P_1 \bowtie \ldots \bowtie P_n$ and returns each result that instantiates all atoms of Q. Our query answering scheme thus requires distributed computation only for answers spanning partition boundaries.

4.2 Identifying Redundant Answers

Checking whether some $\mu_1 \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$ is redundant for Q and i requires one to consider each $\mu \in \mathsf{ans}(Q, G)$, which is clearly impractical. Thus, in this section we present an approximate redundancy check. Note that Theorem 1 holds even if some μ_i in Condition (2a) is redundant, so using an approximate check is safe from the correctness point of view.

Our optimisation is based on the fact that our data partitioning scheme ensures that answers to subject–subject joins are always local. Hence, if, for some μ, each 'star' in Q contains an atom that is not valid for μ, then μ is redundant. This is captured formally in Proposition 1; its proof is trivial, so we omit it for the sake of brevity.

Proposition 1. *Consider an arbitrary $1 \leq i \leq n$ and an arbitrary variable assignment $\mu \in \mathsf{ans}([Q]_{V_i}, [G]_{V_i})$. Then, μ is redundant for Q and i if, for each term $s \neq *$ occurring in $\mu([Q]_{V_i})$ in a subject position, an atom $A \in Q$ exists such that s occurs in the subject position of $\mu([A]_{V_i})$ and $* \in \mathsf{voc}(\mu([A]_{V_i}))$.*

4.3 Limitations of our Query Answering Strategy

Practical applicability of our approach depends critically on effective removal of redundant answers. As we show in Section 5, the optimisation from Proposition 1 is effective on some, but not on all queries. The latter is often the case for long chain queries (i.e., queries of the form $\langle x_0, R_1, x_1\rangle \wedge \ldots \wedge \langle x_{n-1}, R_n, x_n\rangle$): in each

partition element, the wildcard resource typically has a large fan-out and fan-in, and it also occurs in triples of the form $\langle *, R_i, * \rangle$, which can give rise to a large number of answers that are not redundant.

As a possible remedy, we shall explore the possibility of adapting the approach presented in [21]. We envisage an algorithm that evaluates a query in each partition element using nested index loop joins; however, as soon as the algorithm matches some variable to $*$, the algorithm sends the variable matches identified thus far to other servers for continued evaluation. Such an algorithm would still produce all local answers locally and without breaking the query up into pieces, thus reaping the same benefits as the the approach we presented in this paper, but it would not explore any redundant answers. The main open question is to develop a suitable query planning algorithm.

5 Experimental Evaluation

In this section we experimentally evaluate our approach using the Lehigh University Benchmark (LUBM) and the SPARQL Performance Benchmark (SP2B). Each test dataset was split into a partition of size 20, and we used the queries available in the respective benchmarks. This size was chosen to make it directly comparable to related works such as [9, 11]. For a fixed dataset, increasing the partition size is likely to increase the number of non-local answers as the data becomes more fragmented; in contrast, fixing partition size while increasing the size of the dataset is likely to reduce the proportion of non-local answers. The extent to which these changes affect partition quality is out of the scope of this paper and we leave it for our future work. As we have already mentioned, we have not yet implemented a complete system that would allow us to measure end-to-end query answering times; hence, we only conducted the following experiments.

For each $\mathbf{G} = (G_1, \ldots, G_{20})$ of a test dataset G, we calculated (i) the percentage of local answers to test queries, (ii) the storage overhead—that is, the percentage $\frac{|G_1|+\ldots+|G_{20}|-|G|}{|G|}$, and (iii) the number of partial matches to test queries, according to Proposition 1. While experiment (i) determines how many non-local answers must be constructed, experiment (iii) provides us with an indication of how much work is required for this construction. This is critical because, in order to ensure the completeness of query answers, all partial matches in all partition elements must be computed and joined together.

We compared our approach with subject-based hash partitioning (written *Hash*) as in [8, 21], and semantic hash partitioning (written *SHAPE*) [11], which uses an optimised form of subject hashing and directed 2-hop duplication. We did not consider the graph partitioning approach by [9] because SHAPE was shown to offer superior performance. All of these partitioning approaches ensure that all answers to all star queries are local. Furthermore, Proposition 1 ensures there are no partial matches to star queries so we did not consider them in our tests. We used the RDFox system[1] to compute non-local answers, and so we use *RDFox* as the name of our approach.

[1] http://www.cs.ox.ac.uk/isg/tools/RDFox/

5.1 Test Datasets

The Lehigh University Benchmark (LUBM) [6] is a commonly used Semantic Web benchmark. It consists of a synthetic data generator for a simple university domain ontology, and 14 test queries, nine of which are star queries. The generator is parameterised by a number of universities, for which it creates data from the university domain. We used LUBM-2000, containing approximately 267 million triples. The main drawback of LUBM is that the data for each university is highly modular: entities in each university contain many more links amongst themselves than to entities in other universities. We used the five non-star benchmark queries and the following manually created circular query Qc:

```
SELECT DISTINCT ?w ?x ?y ?z WHERE {
    ?x ub:worksFor ?y . ?y ub:subOrganizationOf ?z .
    ?w ub:undergraduateDegreeFrom ?z . ?w ub:advisor ?x }
```

Some LUBM queries have non-empty results only if the data is extended according to the axioms from the LUBM ontology; however, since distributed reasoning is out of scope of this paper, we rewrote the test queries into unions of conjunctive queries in order to take the ontology axioms into account.

The SPARQL Performance Benchmark (SP2B) [16] is another synthetic benchmark that produces DBLP-like bibliographic data. We used an SP2B dataset with approximately 200 million triples. The benchmark provides 12 queries, of which we have used the five non-star queries for our comparison. Some of these queries contain OPTIONAL clauses, which we simply deleted because optional matches are currently not supported in our framework.

5.2 Partition Quality

Table 1 shows the percentage of local answers for each LUBM query. RDFox and SHAPE were able to answer all queries completely, which is in part due to the modular nature of the data; however, hashing performs poorly on all queries. Table 2 shows the results for SP2B. Again, hashing performs very poorly. Furthermore, both RDFox and SHAPE handled queries 4 and 6 well; however, RDFox significantly outperformed SHAPE on queries 5, 7, and 8.

One can intuitively understand these results as follows. Hashing by subject, although effective for star queries, performs very poorly for other types of query: in most cases, it provides almost no local answers. Thus, hashing is likely to be a poor data partitioning scheme for applications with diverse query loads. SHAPE considerably improves hashing, to the extent that only two benchmark queries are problematic. However, by partitioning the data based on its structure, one can further improve the overall performance: our approach is weakest on query Q5 from SP2B, but it still provides a high percentage of local answers.

Table 1: LUBM Percentage of Local Answers

System	Q2	Q8	Q9	Q11	Q12	Qc
RDFox	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
SHAPE	100.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Hash	0.44%	4.96%	0.23%	5.80%	0.00%	0.04%

Table 2: SP2B Percentage of Local Answers

System	Q4	Q5	Q6	Q7	Q8
RDFox	95.95%	73.00%	99.90%	92.41%	91.45%
SHAPE	95.23%	9.72%	100.00%	41.97%	73.72%
Hash	0.01%	0.77%	0.25%	0.08%	0.26%

Table 3: Storage Overhead

	RDFox	SHAPE	Hash
LUBM	3.60%	84.23%	0.00%
SP2B	0.60%	38.63%	0.00%

5.3 Storage Overhead

As we have already discussed, the percentage of local answers can be increased using n-hop duplication, but at the expense of storage overhead. For example, with 2-hop duplication, the approach by [9] can incur an overhead up to 430%.

Table 3 shows the overhead for all partitioning schemes and data sets we considered in our experiments. Hashing clearly incurs no overhead; moreover, although SHAPE incurs a considerable overhead, that can be acceptable for some applications. Our partitioning scheme, however, exhibits a negligible overhead. Intuitively, this is due to the fact that min-cut graph partitioning tries to minimise the number of cut edges, which leads to a small level of duplication.

5.4 Query Evaluation

We evaluated each test query on each partition element, and we discarded all valid assignments and some redundant assignments (according to Proposition 1). For each query, we computed the mean, minimum, maximum, and the sum of the numbers of partial matches across all partition elements.

On LUBM, queries 2, 8, 11 and 12 had no partial matches, so they can be evaluated fully locally without the need for any distributed processing. Queries 9 and c had 6 and 11, respectively, partial matches in total, so the necessary distributed processing is negligible.

On SP2B, evaluating queries 4 and 8 on all partition elements did not finish within an hour, producing very large numbers of partial matches. Since the number of partial matches in each partition element is bounded by the number of non-local answers and the latter is small (cf. Table 2), this result shows that Proposition 1 is not very efficient in identifying redundant answers for queries 4 and 8. Table 4 summarises the results for the remaining queries; in order to better understand these numbers, the table also shows the numbers of total and non-local answers. For queries 5 and 7, the numbers of partial matches are much smaller than the numbers of non-local answers, suggesting that joining the partial matches should be practically feasible. In contrast, the number of partial

Table 4: Partial Matches for SP2B

Query	Total Answers	Non-local Answers	Partial Matches			
			Mean	Min	Max	Total
Q5	2,970,234	801,958	19,128	1,847	43,755	382,564
Q6	38,111,881	38,048	819,709	117,781	1,321,781	16,394,172
Q7	184,193	13,989	2,874	642	6,528	57,471

matches to query 6 is orders of magnitude larger than the number of non-local answers, suggesting that joining the partial answers might be difficult.

To summarise, our approach produces no or few partial matches on many types of query, but it runs into problems with long chain queries such as SP2B query 8. We shall try to improve on this using the ideas outlined in Section 4.3.

6 Conclusion

We have presented a new scheme for partitioning RDF data across a cluster of shared-nothing servers. Our main goal is to minimise the number of connections between partition elements so as to ensure that most answers to typical queries are local (i.e., they can be obtained by evaluating the query locally in all partition elements). We encode in each partition element links to other partition elements, and we use this information in a novel query answering scheme to correctly compute all answers to queries. Unlike existing systems, our query answering scheme retrieves all local answers by simply evaluating the query in each partition element, and it uses the encoded links to reduce the need for distributed processing. We have shown that, on the LUBM and SP2B benchmarks, test queries have more local answers under our data partitioning scheme than with subject-based hashing or semantic partitioning [11], and that our data partitioning scheme incurs a negligible storage overhead. Finally, we have shown that our query answering scheme is effective on many, but not all test queries.

We see two main challenges for our future work. First, we shall try to adapt the graph exploration technique by [21] to obtain a more robust query answering scheme. Second, we shall extend the RDFox system to a fully fledged distributed RDF data store and compare it with existing systems.

Acknowledgements Our work was supported by a doctoral grant by Roke Manor Research Ltd, an EPSRC doctoral training grant, and the EPSRC project MaSI^3.

References

1. Abadi, D.J., Marcus, A., Madden, S.R., Hollenbach, K.: SW-Store: A Vertically Partitioned DBMS for Semantic Web Data Management. The VLDB Journal (2009)

2. Carroll, J.J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K.: Jena: implementing the semantic web recommendations. In: WWW (Alternate Track Papers & Posters) (2004)
3. Dean, J., Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters. Commun. ACM (2008)
4. Erling, O., Mikhailov, I.: Virtuoso: RDF Support in a Native RDBMS. In: Semantic Web Information Management. Springer Berlin Heidelberg (2010)
5. Giménez-García, J.M., Fernández, J.D., Martínez-Prieto, M.A.: MapReduce-based Solutions for Scalable SPARQL Querying. Open Journal of Semantic Web (OJSW) (2014)
6. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Web Semantics (2005)
7. Harris, S., Lamb, N., Shadbol, N.: 4store: The Design and Implementation of a Clustered RDF Store. In: Proc. of the 5th Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009). Washington DC, USA (October 26 2009)
8. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: A Federated Repository for Querying Graph Structured Data from the Web. In: Aberer, K., Choi, K.S., Noy, N.F., Allemang, D., Lee, K.I., Nixon, L.J.B., Golbeck, J., Mika, P., Maynard, D., Mizoguchi, R., Schreiber, G., Cudré-Mauroux, P. (eds.) Proc. of the 6th Int. Semantic Web Conf. (ISWC 2007). LNCS, vol. 4825, pp. 211–224. Busan, Korea (November 11–15 2007)
9. Huang, J., Abadi, D.J., Ren, K.: Scalable SPARQL Querying of Large RDF Graphs. In: Proceedings of the VLDB Endowment (2011)
10. Karypis, G., Kumar, V.: A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing (1999)
11. Lee, K., Liu, L.: Scaling Queries over Big RDF Graphs with Semantic Hash Partitioning. In: Proceedings of the VLDB Endowment (2013)
12. Neumann, T., Weikum, G.: RDF-3X: a RISC-style Engine for RDF. In: Proceedings of the VLDB Endowment (2008)
13. Owens, A., Seaborne, A., Gibbins, N., Schraefel, M.C.: Clustered TDB: A Clustered Triple Store for Jena. http://eprints.ecs.soton.ac.uk/16974/
14. Papailiou, N., Konstantinou, I., Tsoumakos, D., Koziris, N.: H2RDF: adaptive query processing on RDF data in the cloud. In: WWW (2012)
15. Rohloff, K., Schantz, R.E.: High-performance, Massively Scalable Distributed Systems Using the MapReduce Software Framework: The SHARD Triple-store. In: Programming Support Innovations for Emerging Distributed Applications (2010)
16. Schmidt, M., Hornung, T., Lausen, G., Pinkel, C.: SP2Bench: A SPARQL Performance Benchmark. CoRR (2008)
17. Shao, B., Wang, H., Li, Y.: The Trinity graph engine. Tech. rep., Microsoft Research (2012)
18. Sun, J., Jin, Q.: Scalable RDF Store Based on HBase and MapReduce. In: International Conference on Advanced Computer Theory and Engineering (2010)
19. Urbani, J., Kotoulas, S., Maassen, J., van Harmelen, F., Bal, H.E.: WebPIE: A Web-scale Parallel Inference Engine using MapReduce. Journal of Web Semantics 10, 59–75 (2012)
20. Weiss, C., Karras, P., Bernstein, A.: Hexastore: Sextuple Indexing for Semantic Web Data Management. In: Proceedings of the VLDB Endowment (2008)
21. Zeng, Z., Yang, J., Wang, H., Shao, B., Wang, Z.: A Distributed Graph Engine for Web Scale RDF Data. In: Proceedings of the VLDB Endowment (2013)