On Enhancing Scalability for Distributed RDF/S Stores

George Tsatsanifos
School of Electrical & Computer Engineering
National Technical University of Athens
gtsat@dblab.ece.ntua.gr

Dimitris Sacharidis
Institute for the Management of Information Systems
R.C. “Athena”
dsachar@imis.athena-innovation.gr

Timos Sellis
Institute for the Management of Information Systems
R.C. “Athena”
timos@imis.athena-innovation.gr

ABSTRACT
This work presents MIDAS-RDF, a distributed P2P RDF/S repository that is built on top of a distributed multi-dimensional index structure. MIDAS-RDF features fast retrieval of RDF triples satisfying various pattern queries by translating them into multi-dimensional range queries, which can be processed by the underlying index in hopes logarithmic to the number of peers. More importantly, MIDAS-RDF utilizes a labeling scheme to handle expensive transitive closure computations efficiently. This allows for distributed RDFS reasoning in a more scalable way compared to existing methods, as also demonstrated by our extensive experimental study. Furthermore, MIDAS-RDF supports a publish-subscribe model that enables remote peers to selectively subscribe to RDF content.

Categories and Subject Descriptors
H.2.4 [Database Management]: Systems—Distributed databases

General Terms
Algorithms

Keywords
RDF/S stores, P2P systems

1. INTRODUCTION
The main incentive for the Semantic Web is to allow the description of web resources through a formal language, e.g., creating metadata according to a formal representation, forming a mesh of information linked up in such a way as to be easily processable by machines, on a global scale. The Semantic Web community has developed a set of standards for expressing schemas that enable the creation and exchange of information among communities of different backgrounds (e.g., cultural heritage researchers, librarians, audio-visual content producers), and also facilitate the encoding, exchange, processing and reuse of resource metadata while each user community is free to specify its own description semantics in a standardized, interoperable human-readable manner.

The most prominent Semantic Web technology is the Resource Description Framework (RDF) data model, according to which information is represented as statements about resources. An RDF statement is represented as the triple (subject, predicate, object), which signifies that the relationship denoted as predicate holds between the concepts denoted as subject and object, where predicate and object are resources or strings. The power and versatility of this model lies in its simplicity; e.g., relational and XML databases can be described by RDF triples. More importantly, the RDF data model offers the possibility to derive new knowledge from explicit and background knowledge.

Efficient and scalable management of RDF triples has become an important prerequisite for realizing the Semantic Web vision. Centralized RDF stores are empowered with traditional relational tools, e.g., indices, materialized views, to allow for efficient retrieval of RDF triples, and generation of new knowledge via reasoning rules. Motivated by the need to increase flexibility and ultimately allow any community to freely contribute its knowledge, a trend towards decentralized approaches has recently emerged. However, existing solutions fail to offer the scalability of their centralized counterparts. One reason is that single dimensional indices are used in distributed repositories, while RDF triples are inherently multi-dimensional. Moreover, existing algorithms for distributed reasoning based on transitive relationships must visit in sequence a large number of peers.

This work proposes a novel distributed RDF store, called MIDAS-RDF, that alleviates the problems typically encountered in existing solutions. Specifically, our store is based on a structured multi-dimensional index [38], which is able to process point and range queries with low latency, requiring only logarithmic, in the network size, number of hops. Building upon this functionality, MIDAS-RDF supports pattern queries, conjunctive and disjunctive queries, and transitivity queries. The most prominent feature of MIDAS-RDF, however, is that it implements a rigorous inference model that allows efficient identification and storage of new knowledge, by incorporating transitivity information using a labeling scheme. In addition, we propose an RDF publish-subscribe model that enables peers to selectively subscribe to RDF content. Unlike most distributed solutions that are restricted to specific topic-based subscriptions, MIDAS-RDF support general content-based subscriptions.
The remainder of this paper is organized as follows. Section 2 reviews relevant literature about distributed RDF repositories, introduces essential preliminaries, and overviews the underlying distributed index used in MIDAS-RDF. Section 3 describes our storage model and discusses RDF triple retrieval, and Section 4 presents the distributed inference algorithms. Section 5 discusses our decentralized publish-subscribe mechanism. Section 6 contains the experimental evaluation, while Section 7 concludes this work.

2. BACKGROUND AND RELATED WORK

This section establishes the necessary background, reviews literature on distributed RDF stores and RDFS reasoning, and overviews the MIDAS distributed index.

2.1 RDF/S Concepts

The Semantic Web [14] is a group of methods and technologies to allow machines to understand the meaning — or “semantics” — of information on the World Wide Web. The term was coined by World Wide Web Consortium (W3C) [4] director Tim Berners-Lee, who defines the Semantic Web as “a web of data that can be processed directly and indirectly by machines”. These technologies include the Resource Description Framework (RDF), a variety of data interchange formats (e.g., RDF/XML, N3, Turtle, N-Triples), and notations such as RDF Schema (RDFS) and the Web Ontology Language (OWL), all of which are intended to provide a formal description of concepts, terms, and relationships within a given knowledge domain.

The Resource Description Framework (RDF) data model [24] represents information as statements about resources. Any concept that can have a Universal Resource Identifier (URI) can be a resource. RDF consists of a set of named binary predicates, termed RDF statements, that are represented as triples. A (subject, predicate, object) triple implies that the relationship denoted as predicate holds between the concepts denoted as subject and object, where predicate and object are resources or strings.

A set of RDF triples defines an RDF graph. The set of nodes is the set of subjects and objects of the triples. A (u, v) triple defines a directed edge from the subject u to the object v labelled with the property α. Figure 1a shows an example RDF graph. The (Picasso, paints, Guernica) triple is represented as the edge labelled paints from node Picasso to Guernica. The set of all triples regarding a property α define the α subgraph of the RDF graph. For instance, Figure 1b depicts the subgraph for property sc. In a subgraph, a node u subsumes another v if there is a path from v to u. In Figure 1b, Artist subsumes Sculptor, Painter and Cubist.

The RDF specification includes a mechanism, termed the RDF Schema (RDFS), that provides a type system for RDF models. It offers flexible features for representing domain knowledge. Abstraction is enabled by multiple class inheritance with the rdfs:subClassOf and rdfs:subPropertyOf core properties (abbreviated as sc and sp in Figure 1), and classification of resources rdf:type (abbreviated as type). Although RDFS does not provide elaborate mechanisms for defining property restrictions, one can declare simple domain and range restrictions through properties rdfs:domain and rdfs:range (abbreviated as dom and ran in Figure 1).

The semantics of RDFS is defined through a set of axiomatic triples and entailment rules [3], which determine the full set of valid inferences. These inference rules can be intuitively explained as follows. Each rule has a set of premises that conjunctively define its body. The premises represent “extended” RDF statements, where variables can occupy any of the three possible positions in the triple (that of a subject, of a predicate, or of an object). The head of the rule comprises of one or more consequences, each of which represents in its turn an RDF statement. Applying the entailment rules to RDF graphs infers new triples, which can be regarded as a logical consequence of the initial graph. The newly inferred triples can be denoted as inferred closure of the source graph. The only way to determine whether a specific statement can be inferred from an RDF graph is to check if it is a member of its inferred closure.

In simple rule-based systems, there are two main reasoning strategies. Forward-chaining involves applying the inference rules to the known facts to generate new facts. The rules can then be re-applied to the combination of original and inferred facts to produce yet more new facts. The process is iterative and continues until no new facts can be generated. The advantage of this approach is that when all inferences have been computed, query answering is extremely fast. The disadvantages are greater initialization costs (inference computed at load time) and space usage (especially when the number of inferred facts is very large).

On the other hand, backward-chaining involves starting with a fact to be proved or a query to be answered. Typically, the reasoner examines the knowledge base to see if the fact to be proved is present and if not it examines the rule set to see which rules could be used to prove it. Hence, a check is made to see what other supporting facts would need to be present to fire these rules. The reasoner searches for proofs of each of these supporting facts in the same way and iteratively maps out a search tree. The process terminates when either all of the leaves of the tree have proofs or no new candidate solutions can be found. Query processing is similar, but only stops when all search paths have been explored. The purpose in query answering is to find not just one, but all possible substitutions in the query expression. The advantages of this approach are that there are no inferring costs at start-up and minimal space requirements. The disadvantage is that inference must be done each and every time a query is answered and for complex search graphs this can be computationally expensive and slow.

The prevalent language to query RDF data is SPARQL [2]. Similar to SQL in its syntax, it can be used to express queries across diverse data sources, whether the data is stored natively as RDF or viewed as RDF via middleware. It can also query required and optional graph patterns along with their conjunctions and disjunctions.

Figure 1: An RDF graph example
RDF stores, e.g., Hexastore [40], RDFSuite [10], 3store [27], DLDB [32, 33], KAON [39], are centralized engines, similar to the Database Management Systems (DBMS), that allow storing, querying, managing and reasoning on RDF graphs. The architectures enabling RDF stores include giant triple table, property tables, vertical partitioning [5, 19] and application specific indices.

2.2 Distributed RDF Storage and Reasoning

Table 1 overviews the features of the most well-known distributed RDF stores and MIDAS-RDF. Edutella [30] retains the structure of many independent RDF databases and connects them by an unstructured overlay network consisted of peers and super-peers. All data remain at their original location and queries are routed via flooding. Nonetheless, there is no guarantee that the queried content will eventually be retrieved, if at all. Compared to this work, MIDAS-RDF relies on a structured multi-dimensional distributed index that is known to support efficiently range search.

Crespo and Molina argue [20] that a classification hierarchy, resembling an ontology, should serve as the basis of the network formation. In Semantic Overlay Networks (SONs), nodes of semantically similar content are clustered together, establishing connections among them; a node is allowed to belong to more than one clusters at the same time. Queries are processed by identifying which SONs are better suited to answer it. Then, the query is sent to a node in those SONs to multi-cast it to the remaining members. On the other hand, MIDAS-RDF does not require a notion of semantic proximity and features efficient routing mechanisms.

RDFPeers [15] is one of the first efforts for structured peer-to-peer RDF stores. The key idea is to use a MAAN overlay [16] to index a triple three times, once based on the subject, another based on the predicate, and a final based on the object. There are a few shortcomings in this design. First, there is a replication factor of three for all triples. Furthermore, the RDF schema semantics are totally ignored during query routing. Due to its underlying storage layer, RDFPeers can in the worst case, e.g., for queries with low selectivity, require a linear number of hops with respect to the overlay size. Finally, RDFPeers is susceptible to load imbalances, as peers responsible for popular and reserved keywords quickly become overwhelmed. Note that MIDAS-RDF does not store duplicate triples, has efficient routing algorithms with logarithmic number of hops in the worst case, does not suffer from load balances due to popular keywords, and in addition supports reasoning.

Regarding distributed RDFS reasoning in particular. Fang et al. [23] propose an iterative forward-chaining procedure, though they do not address load-balancing issues. Kaoudi et al. [28] compare the two well-known approaches for RDFS reasoning, backward- and forward-chaining on top of distributed hash table based structure overlays, and conclude that backward-chaining is the most effective. Compared to this work, MIDAS also considers incremental updates, triple removals along with all their inferred triples, and prevents the inference of duplicate triples. Battré et al. [12] present a paradigm for performing reasoning over locally stored triples and introduce a policy to address load-balancing issues. On the other hand, our approach does not require from each peer to maintain multiple RDF databases for distinct purposes. Marvin [31] constitutes a parallel and distributed platform for processing large amounts of RDF data, on a network of loosely coupled peers. Also, this work presents an iterative procedure for computing the deductive closure of large datasets. Serafini and Tamarin propose in [36] a system that relies on manually created ontology mappings.

A different concept is presented in [35] that distributes the reasoning rules instead of the data, so that each node is responsible for performing a specific part of the reasoning process. However, this scheme is liable to imbalances and does not scale well. Piazza [26] relies on the mappings established between the individual peer schemes to route queries on semantically related peers, rather than on a distributed index of graph fragments. GridVine [7] realizes semantic overlays by separating a logical layer from a physical, applying the principle of data independence. The logical layer has operations to support semantic interoperability including attribute based search, schema inheritance, schema management and schema mapping. It also supports a schema reconciliation technique, known as semantic gossiping, for semantic interoperability in decentralized settings [6].

2.3 Labeling Schemes

In the following, we provide a succinct overview of labeling schemes for the Semantic Web; a detailed review can be found at [18] and [17]. These techniques can be divided into two categories, interval-based and prefix-based.

Agrawal et al. [9] propose an scheme that assigns intervals to nodes of a graph, so that an ancestry relationship between two nodes is checked by interval inclusion. First, spanning tree T is determined. A node u in T is labeled with \[\text{minpost}(u), \text{post}(u)\], where \(\text{post}(u)\) is the order of u in a postorder traversal of T, and \(\text{minpost}(u)\) is the lowest post among u’s descendants. Finally, all nodes of the graph are examined in reverse topological order, and for each edge \((u, v)\) all intervals associated with \(v\) are propagated to \(u\). Based on this scheme, a node \(v\) is an ancestor of \(u\), if all intervals of \(u\) are included in those of \(v\). Note that to avoid frequent node relabeling and support incremental updates, gaps are typically left in the generated intervals.

Other interval schemes exhibit similar properties. In [21], a node \(u\) in \(T\) is labeled with \([\text{pre}(u), \text{post}(u)]\), where \(\text{post}(u)\) is the order of \(u\) in a preorder traversal of \(T\). Similarly, Tsakalidis [37] shows that a node \(v\) is an ancestor to \(u\) iff \(\text{pre}(v) \leq \text{pre}(u) \leq \text{post}(v)\), which leads to a labeling scheme with labels of size \(2 \log n\). Peleg [34] proposes an \(O(\log n)\) labeling scheme that given nodes \(u\) and \(v\) one can determine the lowest common ancestor of \(u\) and \(v\).

Prefix-based schemes, such as [29], [11], [8], apply for trees. Consider an alphabet \(\Sigma = \{\sigma_1, \ldots, \sigma_D\}\). Node labels can be defined recursively, as follows. Consider a node \(u\) with label \(\text{id}(u) \in \Sigma^*\). Assuming an order on \(u\)’s children, let node \(v\) be the \(k\)-th child of \(u\). We have that \(\text{label}(v) = \text{label}(u)\sigma_k\). One of the advantages of this approach is its ability to handle incremental updates efficiently. As long as the order among descendants is not important, one can always add a new child as the last, avoiding any relabeling. Note that to handle the general case of graphs, a child is allowed to have multiple labels, one per parent.

Checking whether a node \(v\) is an ancestor of \(u\) is equivalent to checking if \(\text{label}(v)\) is a prefix of \(\text{label}(u)\). Furthermore, given two nodes \(u\) and \(w\), their nearest common ancestor is the node labeled with their longest common prefix, which can be easily computed in \(O(\min[|\text{label}(u)|, |\text{label}(w)|])\), where \(|\text{label}(u)|\) denotes the length of \(u\)’s identifier.
Table 1: Characteristics of different distributed RDF repositories.

### 2.4 The MIDAS Overlay

We present a brief overview of the MIDAS distributed multi-dimensional index [38], which forms the basis for our distributed RDF store. MIDAS is a distributed version of the k-d tree [13]. The k-d tree is a binary tree, where each node corresponds to an axis parallel rectangle; the root represents the entire space. Each internal node has always two children whose rectangles are obtained by splitting the parent’s rectangle in two along some dimension. Each node is assigned a binary identifier corresponding to its path from the root, defined recursively. The root has the empty id \( \emptyset \); the left (resp. right) child of an internal node has the id of its parent augmented with 0 (resp. 1). Figure 2 depicts a k-d tree of seven nodes with three splits and their ids. Due to the hierarchical splits, the rectangles of the leaf nodes in a k-d tree, shown in green, constitute a non-overlapping partition of the entire space.

![Figure 2: An example of a two-dimensional k-d tree.](image)

A peer in MIDAS corresponds to a leaf of the k-d tree, and stores/indexes all tuples that reside in the leaf’s rectangle. Peer joins and departures are handled by appropriately splitting and merging k-d tree nodes. Due to its routing mechanism and the locality preserving property of the underlying k-d tree, MIDAS can efficiently support multi-dimensional point and range queries. A point in the space (more accurately, the peer responsible for that point) can be retrieved in logarithmic number of hops, in the worst case. To process range queries, MIDAS is able to identify in parallel all peers whose rectangle overlaps with the given range. As a result, independently of the size of the range, MIDAS requires logarithmic number of hops in the worst case.

### 3. THE MIDAS-RDF DISTRIBUTED STORE

Section 3.1 describes how triples are stored and indexed, while Section 3.2 explains query evaluation in MIDAS-RDF.

3.1 Storage Model

The MIDAS-RDF system stores RDF triples and transitive information using the MIDAS distributed index. An RDF triple \( (u, \alpha, v) \) is represented as a four dimensional key \( \langle u, \alpha, v, \text{id}(v) \rangle \), where \( \text{id}(v) \) encodes transitive information regarding object \( v \) in the subgraph of predicate \( \alpha \), which corresponds to the RDF graph associated with the triples with predicate \( \alpha \). We also assume a lexicographic ordering for the \( \text{subject}, \text{predicate} \) and \( \text{object} \) dimensions, and the natural ordering for \( \text{object subgraph identifiers} \).

Consider a MIDAS-RDF key \( \langle u, \alpha, v, \text{id}(v) \rangle \). The subgraph identifier of the object \( v \), \( \text{id}(v) \), can be set according to one of the schemes described in Section 2.3. When the interval-based scheme of [9] is used, \( \text{id}(v) \) is \( \text{post}(v) \), that is, \( v \)'s order in a postorder traversal of a spanning tree of the \( \alpha \) subgraph. On the other hand, when the prefix-based scheme of [29] is used, \( \text{id}(v) \) corresponds to the prefix label \( \text{label}(v) \) in a spanning tree of the \( \alpha \) subgraph that includes the \( (u, v) \) edge.

Note that, when the interval-based scheme is used, \( \text{post}(v) \) alone cannot encode the transitivity in the \( \alpha \) subgraph. For this reason, with each \( \langle u, \alpha, v, \text{post}(v) \rangle \) key, MIDAS-RDF associates a composite value that consists of all intervals associated with node \( v \) in the \( \alpha \) subgraph. Therefore, in this case, a tuple in MIDAS-RDF has five dimensions, \( \text{subject}, \text{predicate}, \text{object}, \text{object subgraph identifier}, \) and \( \text{intervals} \). However, only the first four comprise the key indexed by MIDAS-RDF.

Figure 3 illustrates an example subgraph of some property that contains six nodes denoted symbols \( u \) through \( z \); we assume that other nodes below \( x, y, z \) exist, but are not shown. For the interval-based scheme, the spanning tree contains all edges except the one drawn with dotter line. For easy reference, the node symbols also correspond to the identifiers according to the interval-based scheme, e.g., \( \text{post}(u) = u \). The intervals associated with each node are shown in red. Observe that node \( w \) has two intervals, one from the spanning tree edge \( [r, w] \), and another \( [q, y] \), propagated from node \( y \). The identifiers according to the prefix-based scheme are depicted on the edges and are shown in blue. Notice that node \( y \) has two prefix labels, 01 corresponding to the \((v, y)\) edge, and 10 corresponding to the \((w, y)\) edge.

Table 2 shows the attribute values of the tuples stored in MIDAS-RDF peers according to both labeling schemes. In this example, the predicate dimension is omitted since its common among all triples. In the prefix-based scheme, columns 1, 2 and 3 constitute the tuple keys, whereas in the interval-based scheme, columns 1, 2 and 4 constitute the tuple keys, and column 5 represents the value.
3.2 RDF Queries

MIDAS-RDF straightforwardly supports RDF triple pattern queries. Consider the query types shown in Table 3. The notation `?s` (and `?p`, `?o`) is borrowed from SPARQL [2], and denotes that variable `?s` corresponds to possibly many subjects satisfying the pattern. Query type Q1 is evaluated in MIDAS-RDF as a point query requiring logarithmic, in the number of peers, hops. All other pattern queries, are evaluated as range queries in MIDAS-RDF, where an `?s`, `?p`, or `?o` expression is translated as the range of the entire domain of subjects, predicates, or objects, respectively. For example, consider the RDF graph depicted in Figure 1a. The Q2 type query SELECT ?paint WHERE {Picasso paints ?paint} returns all paintings of Picasso. Note that, a Q8 type query essentially retrieves all RDF triples. Due to the MIDAS overlay, such queries require only logarithmic hops.

Table 2: Triples stored as tuples for both labeling schemes.

<table>
<thead>
<tr>
<th>Subject</th>
<th>Object</th>
<th>Label</th>
<th>Post</th>
<th>Intervals</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>v</td>
<td>0</td>
<td>v</td>
<td>[p,v]</td>
</tr>
<tr>
<td>u</td>
<td>w</td>
<td>1</td>
<td>w</td>
<td>[r,w], [q,y]</td>
</tr>
<tr>
<td>v</td>
<td>x</td>
<td>00</td>
<td>x</td>
<td>[p,x]</td>
</tr>
<tr>
<td>v</td>
<td>y</td>
<td>01</td>
<td>y</td>
<td>[q,y]</td>
</tr>
<tr>
<td>w</td>
<td>y</td>
<td>10</td>
<td>y</td>
<td>[q,y]</td>
</tr>
<tr>
<td>w</td>
<td>z</td>
<td>11</td>
<td>z</td>
<td>[r,z]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Subject</th>
<th>Predicate</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1</td>
<td>s</td>
<td>p</td>
<td>o</td>
</tr>
<tr>
<td>Q2</td>
<td>s</td>
<td>p</td>
<td>?o</td>
</tr>
<tr>
<td>Q3</td>
<td>s</td>
<td>?p</td>
<td>o</td>
</tr>
<tr>
<td>Q4</td>
<td>s</td>
<td>?p</td>
<td>?o</td>
</tr>
<tr>
<td>Q5</td>
<td>?s</td>
<td>p</td>
<td>o</td>
</tr>
<tr>
<td>Q6</td>
<td>?s</td>
<td>p</td>
<td>?o</td>
</tr>
<tr>
<td>Q7</td>
<td>?s</td>
<td>?p</td>
<td>o</td>
</tr>
<tr>
<td>Q8</td>
<td>?s</td>
<td>?p</td>
<td>?o</td>
</tr>
</tbody>
</table>

Table 3: Atomic triple pattern queries.

Disjunctive range queries can be resolved by issuing one query for each contiguous range and then computing the union of the results. Such a mechanism can be efficiently implemented by issuing simultaneously the atomic queries that constitute the disjunction. Conjunctive queries, such as SELECT ?artifact WHERE {(?x type Sculptor) (?x type Painter)} {?x creates ?artifact} returns the artifacts that were created by an artist that was a painter and a sculptor, e.g., Michelangelo’s David. A simple way to support conjunctive queries is to invoke first the atomic query that constitutes the conjunction with the greater selectivity (assuming that such information exists, e.g., via histograms). Then, the result-set can be filtered appropriately in the peer that issued the request.

An important class of queries is those concerning transitive properties. For example, consider the sc subgraph of Figure 1b. A query for the concepts that subsume Cubist would retrieve Painter and Artist. Efficient processing of transitivity queries requires a labeling scheme. Posed such a query, MIDAS-RDF initially retrieves v’s object subgraph identifier id(v), if not known. Further, if the interval-based scheme is used, it also retrieves the labels associated with v. Based on this information it formulates a range query.

In the case of the interval-based scheme, a range query is issued for its interval associated with v. For example, the nodes (concepts) that subsume v in Figure 3 have ids in the range [0, 1]; these are nodes x and y.

In the case of the interval-based scheme, a range query is issued for its interval associated with v. For example, the nodes that subsume v in Figure 3 have ids that belong in the range [p, v]. Note that, independently of the labeling scheme, transitivity queries are processed in logarithmic number of hops, as they are transformed into range queries.

4. RDFS REASONING IN MIDAS-RDF

Section 4.1 outlines the basic principles of the MIDAS-RDF inference model. Then, Section 4.2 details the implementations of RDFS entailment rules.

4.1 Inference Model

The RDF semantic document [3] describes how RDFS entailment can be seen as a set of rules which generate new RDF triples from existing ones. In this work, we focus on a subset of these rules, including the extensional entailment rules, depicted in Table 4, which contains the most computationally intensive ones. Properties sp and sc are shortcuts for subPropertyOf and subClassOf, respectively.

Table 4: RDFS entailment rules.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Precondition</th>
<th>Generated Triple</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs2</td>
<td>(α, domain, x), (u, α, v)</td>
<td>(u, type, x)</td>
</tr>
<tr>
<td>rdfs3</td>
<td>(α, range, x), (u, α, v)</td>
<td>(u, type, x)</td>
</tr>
<tr>
<td>rdfs5</td>
<td>(α, sp, β), (β, sp, γ)</td>
<td>(α, sp, γ)</td>
</tr>
<tr>
<td>rdfs7</td>
<td>(α, sp, β), (u, α, v)</td>
<td>(u, β, v)</td>
</tr>
<tr>
<td>rdfs9</td>
<td>(u, sc, v), (u, type, u)</td>
<td>(u, type, v)</td>
</tr>
<tr>
<td>rdfs11</td>
<td>(u, sc, v), (v, sc, w)</td>
<td>(u, sc, w)</td>
</tr>
<tr>
<td>ext1</td>
<td>(α, domain, u), (u, sc, v)</td>
<td>(α, domain, v)</td>
</tr>
<tr>
<td>ext2</td>
<td>(α, range, u), (u, sc, v)</td>
<td>(α, range, v)</td>
</tr>
<tr>
<td>ext3</td>
<td>(α, domain, u), (β, sp, α)</td>
<td>(β, domain, u)</td>
</tr>
<tr>
<td>ext4</td>
<td>(α, range, u), (β, sp, α)</td>
<td>(β, range, u)</td>
</tr>
</tbody>
</table>
ture. Naturally, the generated triples for some of the rules may trigger other rules to be executed.

In order to avoid creating duplicates, we adopt the policy that our methods are executed only once from one of the two peers containing the parts to be considered for entailment. Nevertheless, that part is selected to optimize specific factors that affect performance and scalability. Therefore, our design choices minimize the amount of exchanged messages and as a result the consumed bandwidth, which is arguably the most costly resource in a distributed environment.

Most importantly, our scheme enhances performance by utilizing a labeling scheme. For example, without such a scheme, the deductive closure of entailment rule rdfs2 on a subClassOf subgraph of depth K, would require a sequence of at least K operations that cost $O(\log n)$ hops each. On the other hand, the labeling scheme, can process the deductive closure in a single operation of $O(\log n)$ cost that applies the rule to all qualifying nodes of the sub-graph. These nodes are efficiently retrieved by issuing an appropriate range query, as described in Section 3.2. To illustrate this, assume the RDF graph of Figure 1a and consider rule rdfs9. In MIDAS-RDF, a peer can generate triples (Picasso, type, Painter) and (Picasso, type, Artist) in a single step. As a result, the query SELECT ?x WHERE (?x type Artist) would also return the name of Picasso, which otherwise would not be part of the answer.

4.2 Implementation Details

We present the implementation of the rules depicted in Table 4, and also discuss the case of triple updates.

4.2.1 Applying rules rdfs2, rdfs3

Algorithm 1 produces new knowledge using entailment rules rdfs2 and rdfs3. This method is called when a triple of the form $(\alpha, \text{domain}, u)$ or $(\alpha, \text{range}, v)$ is locally inserted. Such a triple must be associated with all triples of predicate $\alpha$ retrieved from remote peers by invoking a range query, so as to generate the triples dictated by the two rules.

The example illustrated in Table 5 concerns rule rdfs2 and can be described by the following steps:

1. Peer $p_2$ examines local triple $(u, \alpha, v)$ and invokes a range query of the form $(\alpha, \text{domain}, ?X)$ to retrieve $(\alpha, \text{domain}, x)$ from $p_1$.
2. Peer $p_2$ processes the returned tuple using rule rdfs2 and generates triple $(u, \text{type}, x)$.
3. Peer $p_2$ stores the generated tuples in the overlay using the network infrastructure. The peers that are responsible for the newly inserted triples will invoke Algorithm 5 to apply rule rdfs9.

Figure 5 also shows that an identical approach for rule rdfs3 applies. Notice that Algorithm 5 handles triples that were just inserted in a peer. In order to prevent the creation of duplicates, we produce new triples when examining a predetermined part (for performance reasons) of the two required to apply an entailment rule. However, as new triples are inferred and inserted there is also the need to implement methods for handling entailment rules for the other part to be considered for entailment. As this is the case for Algorithm 5 and entailment rule rdfs9, we suggest very similar implementations with limited changes for the rules that are not presented both ways, such as rdfs7, ext1 and ext2.

In many ways both implementations are equivalent and our selections were made towards the more efficient and elegant.

<table>
<thead>
<tr>
<th>Peer</th>
<th>Local Store</th>
<th>Generated Triples</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$(\alpha, \text{domain}, x)$</td>
<td>$(u, \alpha, v)$</td>
<td>rdfs2,3</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$(u, \alpha, v)$</td>
<td>$(u, \text{type}, x), (v, \text{type}, z)$</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Inference example for rdfs2 and rdfs3.

Algorithm 2 deduceRDFS2b: Triggered upon an insertion of an RDF triple of the form $(\alpha, \text{domain}, u)$, searches for associated triples by rule rdfs2 and generates new triples to be stored.

1: /* Associated rule: RDFS2 */
2: p.rangeRDFquery ($\beta$, domain, $?X$)
3: for all $x$ in $X$ do
4: p.insert ($u, \text{type}, x$)
5: end for
6:
7: /* Associated rule: RDFS3 */
8: p.rangeRDFquery ($\beta$, range, $?Y$)
9: for all $y$ in $Y$ do
10: p.insert ($v, \text{type}, y$)
11: end for

4.2.2 Applying rules rdfs5, rdfs7

Algorithms 3 and 4 justify the design choice of utilizing a labeling scheme in MIDAS-RDF, as they reduce the number of hops required from $O(K \log n)$ to $O(\log n)$. The example of Algorithm 3 depicted in Table 6 can be described by the following steps:

1. Peer $p_1$ examines $(\alpha, \text{subPropertyOf}, \beta)$ and acquires the local labels of the triples that correspond to the nodes that subsume $\beta$ in the subPropertyOf sub-graph.
2. Peer $p_1$ retrieves all the triples that correspond to the obtained labels in $O(\log n)$ hops, from peers $p_2$ and $p_3$ triples $(\beta, \text{subPropertyOf}, \zeta)$, $(\zeta, \text{subPropertyOf}, \xi)$. 
3. Peer $p_1$ stores the generated tuples, by entailment rule rdfs5, $(\alpha, \text{subPropertyOf}, \zeta)$ and $(\alpha, \text{subPropertyOf}, \xi)$ in the overlay.
4. Likewise, peer $p_2$ checks triple $(\beta, \text{subPropertyOf}, \zeta)$ to retrieve $(\zeta, \text{subPropertyOf}, \xi)$ from $p_3$, and when rule rdfs5 is applied it generates $(\beta, \text{subPropertyOf}, \xi)$ to store it in the overlay.
5. Peer $p_1$ examines $(\zeta, \text{subPropertyOf}, \xi)$ only to find out that there are no nodes that subsume $\zeta$.
6. Peer $p_2$ acquires $(u, \alpha, v)$ (Alg. 3) and applies rule rdfs7 with local labels of the nodes that subsume $\zeta$. 
7. Then, peer $p_2$, after invoking a range query for all relevant labels, retrieves $(\zeta, \text{subPropertyOf}, \xi)$ from $p_3$.
8. When rule $rdfs7$ is applied, peer $p_2$ generates triples $(u, \zeta, v), (u, \xi, v)$ and stores them in the overlay.
9. The peers that are responsible for the newly inserted triples from peer $p_2$ will invoke Algorithm 1 for handling them according to entailment rules $rdfs2$ and $rdfs3$.

<table>
<thead>
<tr>
<th>Peer</th>
<th>Local Store</th>
<th>Generated Triples</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$(\alpha, \text{sp}, \beta)$</td>
<td>$(\alpha, \text{sp}, \zeta), (\alpha, \text{sp}, \xi)$</td>
<td>$rdfs5$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$(\beta, \text{sp}, \zeta)$</td>
<td>$(\beta, \text{sp}, \xi), (u, \zeta, v), (u, \xi, v)$</td>
<td>$rdfs5,7$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$(\zeta, \text{sp}, \xi)$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_4$</td>
<td>$(u, \beta, v)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6: Inference example for $rdfs5$ and $rdfs7$.

Instead of chasing pointers and issuing sequential queries, peer $p_1$ retrieves the subgraph with a simple range query based on a labeling scheme. In particular, peer $p_4$ would have to invoke 4 distinct queries in order to generate all possible triples regarding entailment rule $rdfs7$ for its content. Most importantly, peer $p_4$ is in position of completing the closure of its statements without being aware of other peers’ progress, in our case $p_1, p_2, p_3$, or waiting for their intermediate results.

4.2.3 Applying rules $rdfs11$, $rdfs9$

The example of Algorithm 4 shown in Table 7 is described in the following steps:

1. Peer $p_1$ examines $(u, \text{subClassOf}, v)$ and checks the labels of the nodes that subsume $v$.
2. Peer $p_1$ retrieves triples, $(v, \text{subClassOf}, w)$ from $p_2$ and $(u, \text{subClassOf}, y)$ from $p_3$.
3. Peer $p_1$ stores the generated triples $(u, \text{subClassOf}, w)$ and $(u, \text{subClassOf}, y)$ in the overlay.
4. When peer $p_2$ examines $(v, \text{subClassOf}, w)$, it retrieves from $p_3$ triple $(w, \text{subClassOf}, y)$.
5. Peer $p_2$ stores the generated triple $(v, \text{subClassOf}, y)$ in the overlay.
6. Peer $p_3$ examines $(w, \text{subClassOf}, y)$ only to find out that there are no nodes that subsume $y$. It also issues a range query for $(?X, \text{type}, w)$ and retrieves $(x, \text{type}, w)$ from $p_4$.
7. Peer $p_3$ applies rule $rdfs9$ and generates $(x, \text{type}, y)$ that stores in the overlay.

<table>
<thead>
<tr>
<th>Peer</th>
<th>Local Store</th>
<th>Generated Triples</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$(u, \text{sc}, v)$</td>
<td>$(u, \text{sc}, w), (u, \text{sc}, y)$</td>
<td>$rdfs11$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$(v, \text{sc}, w)$</td>
<td>$(v, \text{sc}, y)$</td>
<td>$rdfs11$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$(w, \text{sc}, y)$</td>
<td>$(x, \text{type}, y)$</td>
<td>$rdfs9$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$(x, \text{type}, w)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Inference example for $rdfs11$ and $rdfs9$.

4.2.4 Applying rules $ext1$ and $ext2$

For the implementation of extentional entailment rules $ext1$ and $ext2$, we follow an approach similar to the one adopted for $rdfs9$. The example shown in Table 8 is described as follows:

1. Peer $p_2$ checks $(u, \text{subClassOf}, v)$ and to apply $ext1$ retrieves the triples that correspond to labels of the nodes that subsume $v$, in our case $(v, \text{subClassOf}, w)$.
2. When entailment rule $ext1$ is applied for each of the returned triples (Alg. 4), $p_2$ generates and stores triples $(\alpha, \text{domain}, v)$ and $(\alpha, \text{domain}, w)$, after retrieving from $p_1$ $(\alpha, \text{domain}, u)$ with a range query.
3. The peers that are responsible for the newly inserted triples will invoke Algorithm 2 for handling $rdfs2$.
4. Likewise, $p_3$ applies $ext2$ to $(\beta, \text{range}, v)$ that retrieves after $O(\log n)$ hops from $p_4$ to produce $(\beta, \text{range}, w)$.

<table>
<thead>
<tr>
<th>Peer</th>
<th>Local Store</th>
<th>Generated Triples</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$(\alpha, \text{domain}, u)$</td>
<td></td>
<td>$ext1$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$(u, \text{sc}, v)$</td>
<td>$(\alpha, \text{domain}, v), (\alpha, \text{domain}, w)$</td>
<td>$ext1$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$(v, \text{sc}, w)$</td>
<td>$(\beta, \text{range}, w)$</td>
<td>$ext2$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$(\beta, \text{range}, v)$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 8: Inference example for $ext1$ and $ext2$.

4.2.5 Applying rules $ext3$ and $ext4$

Entailment rules $ext3$ and $ext4$ require a different approach. All previously described methods resolve a transitive relation whose results were combined with a fixed statement. Here, we compute the transitive closure for each statement we examine and we combine each element of the result with associated remote triples that $ext3$ and $ext4$ require. Next, we provide an intuition of the principle used in Algorithm 3 in the example of Table 9:

1. Peer $p_1$ examines $(\zeta, \text{subPropertyOf}, \beta)$ and uses local information for all labels of the nodes that subsume $\zeta$ in the $\text{subPropertyOf}$ sub-graph.
2. Peer $p_2$ based on the local labeling information invokes the appropriate range query to retrieve statement $(\beta, \text{subPropertyOf}, \alpha)$ from peer $p_2$.
3. Peer $p_3$, for each subsuming node $\alpha$, retrieves all triples of the form $(\alpha, \text{domain}, u)$, in our case from peer $p_1$ and $(\alpha, \text{range}, v)$ from peer $p_4$.
4. When peer $p_3$ applies $ext3$ generates $(\zeta, \text{domain}, u)$ that stores in the overlay. Likewise, peer $p_4$ generates triple $(\zeta, \text{range}, v)$ from $ext4$.
5. The peers that are responsible for the newly inserted triples will invoke Algorithm 2 for handling rules $rdfs2$ and $rdfs3$ for $(\zeta, \text{domain}, u)$ and $(\zeta, \text{range}, v)$.
6. An identical procedure is followed by peer $p_2$ to produce $(\beta, \text{domain}, u)$ and $(\beta, \text{range}, v)$ for $ext3$ and $ext4$.

<table>
<thead>
<tr>
<th>Peer</th>
<th>Local Store</th>
<th>Generated Triples</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$(\alpha, \text{domain}, u)$</td>
<td></td>
<td>$ext3,4$</td>
</tr>
<tr>
<td>$p_2$</td>
<td>$(\beta, \text{sp}, \alpha)$</td>
<td>$(\beta, \text{domain}, u), (\beta, \text{range}, v)$</td>
<td>$ext3,4$</td>
</tr>
<tr>
<td>$p_3$</td>
<td>$(\zeta, \text{sp}, \beta)$</td>
<td>$(\zeta, \text{domain}, u), (\zeta, \text{range}, v)$</td>
<td>$ext3,4$</td>
</tr>
<tr>
<td>$p_4$</td>
<td>$(\alpha, \text{range}, v)$</td>
<td></td>
<td>$ext3,4$</td>
</tr>
</tbody>
</table>

Table 9: Inference example for $ext3$ and $ext4$.

4.2.6 Incremental updates

For a data-driven forward chaining based scheme, an update of triples naturally triggers an update of the inferred
tuples as well. In the process of reverting these changes we need to make sure that all the triples that we have removed have been inferred with one of the RDFS entailment rules. Since there is no notion of data provenance, we follow the reverse procedure of our inference methods with the following additional consideration. Before removing any triple, we must make sure that no other triple exists that when combined under any of RDFS entailment rule would yield the candidate triple. This check can be supported at the cost of some additional steps and computation.

The example of Algorithm 6 in Table 10 applies to deletions of triples of the form ($\alpha$, \text{subPropertyOf}, $\beta$) or (u, $\alpha$, v) from peers p1, p2, which raises the question of removing the (u, $\beta$, v) inferred triple produced by rdfs7. The problem is that the very same triple is inferred by peer p4 when applying rdfs7 to its content. Hence, as more than one combinations of rules and triples may lead to the same inferred rules, MIDAS-RDF must examine whether there are more ways of producing them. If this is the case, we proceed to the next inferred triple. For example, in Table 10 when (u, $\alpha$, v) is deleted and (u, $\beta$, v) is examined we simultaneously issue the appropriate requests that would trace the conflict. First, we have to study all RDFS entailment rules and find those that produce triples of the form (u, $\beta$, v). Then, we will examine whether there are other triples that lead again to (u, $\beta$, v) for each relevant rule that is found. In our case, rdfs7 is the unique rule that needs to be checked. This is done with a range query of the form (u, ?T, v) that retrieves (u, $\gamma$, v) from peer p3. Hence, we now need to check whether node $\beta$ is an ancestor of $\gamma$ in the \text{subPropertyOf} sub-graph, a simple task using a labeling scheme. For the prefix-based scheme, we simply check whether the label of node $\beta$ is a prefix of the label of node $\gamma$. For the interval-based scheme, we check whether the postorder identifier of node $\gamma$ is in any of the intervals of $\beta$. Apparently, the cost of this mechanism is not dramatic if we simultaneously try to resolve the possible conflicts from all relevant rules, as in a MIDAS disjunctive query. Thus, combined with this special feature, the aforementioned methods can easily be undone and all inferred triples be removed by replacing the \text{p.insert()} function-calls, in the algorithms presented earlier, with \text{p.delete()} function-calls. Finally, update requests can be considered as a pair of a deletion of an old triple and a insertion of a triple with the updated values.

5. A PUBLISH-SUBSCRIBE MODEL

In this section, we present a distributed content-based publish-subscribe service that addresses needs pertaining to participants in the community that want to be quickly notified of specific new content, i.e., they have persistent queries expressing interest in certain topics that are constantly serviced. Content-based publish-subscribe systems allow more complex subscriptions by enabling restrictions on the event content. Most importantly, multiple predicates upon the subscription may be defined and only the events that fulfill the requirements are notified to the subscriber.

Our scheme constitutes a middleware for scalable dissemination of data events to subscribers dispersed across the network. In this context, an area of interest is described by a hyper-rectangle, and withah, a range along each dimension.
is defined, to identify triples limited to a specific subject. Dimensions that are of no particular interest, are naturally rendered obsolete by declaring their range of domain values equal to the domain space of that particular dimension.

If a subscription needs to specify multiple predicates over the same attribute (for example conjunctive and disjunctive clauses), we can model such a subscription as a combination of multiple subscriptions, each of which specifies a continuous range over that attribute. Therefore, we treat a subscription request similar to a multi-dimensional range query. Peers may subscribe to a specified area whose responsibility might be distributed among several peers. A subscriber does not know in advance what peers are responsible for the area it is interested in. Moreover, each peer maintains a local spatial index (e.g., an R-tree [25]) with all the subscriptions (hyper-rectangles) that overlap its area of responsibility. To elaborate, as a request is being forwarded recursively to more relevant subtrees of the distributed index, it becomes fragmented by the corresponding sub-regions, and within, the subscription (hyper-rectangle) is stored in the local spatial indices of the overlapping peers. Henceforth, subscribers will be notified about the changes their areas of interest undergo. Consequently, our approach requires an additional hop for insertions/updates to inform the subscribers, as they instantly become aware of changes that take place (e.g., newly inserted triples) in those specific areas. Otherwise, a peer would have to periodically invoke a request of a certain type, which would generate unnecessary load. Note that multiple notifications are prevented as duplicate events can be easily detected in the local index.

6. EXPERIMENTAL EVALUATION

Section 6.1 describes the experimental setting and Section 6.2 presents the results of our extensive evaluation.

6.1 Setting

We evaluate performance according to several metrics appearing. We measure query performance by latency, which is the maximum distance (in terms of hops) from the initial peer to any peer reached during query processing. Another important metric in distributed systems is network congestion, defined as the average number of queries processed at any node, when \( n \) uniformly random queries are issued. This actually resembles the average traffic a peers accepts when \( n \) queries are issued in the overlay by random peers. To quantify how fast an answer is retrieved, we use two metrics. Recall is the ratio between the number of accessed relevant peers to the number of peers that are relevant to the query for each simulation cycle. Responsiveness is the ratio between the number of retrieved qualifying tuples to the number of tuples are relevant.

Our experiments simulate a dynamic environment. They consist of a growing and a shrinking stage. Therefore, we are given the opportunity to study the course of those metrics, as overlay adapts dynamically to changes of the topology. In each step of the growing stage, a new peer joins the overlay network, whereas during the shrinking stage a peer is removed, selected at random with equal probability. In each figure, we depict the impact of both stages. All simulations initiate an overlay of 1K peers growing up to 100K peers, followed by the reverse procedure.

To evaluate transitivity computation and study how different techniques scale, we used synthetic RDF graphs of variable depth (up to 16). In these settings, we compare our labeling scheme based paradigm with traditional solutions based on an iterative forward-chaining procedure. Note that efficient management of transitive relations is of major significance as it is present in the vast majority of the RDFS entailment rules proposed by W3C [9]. The synthetic datasets consist of 1M triples.

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Relationships (in main relationships)</th>
</tr>
</thead>
<tbody>
<tr>
<td>900,440</td>
<td>publication-has-author (author)</td>
</tr>
<tr>
<td>438,531</td>
<td>contained in proceedings (isIncludedIn)</td>
</tr>
<tr>
<td>112,303</td>
<td>cites publication</td>
</tr>
<tr>
<td>10,639</td>
<td>has-homepage (foaf:homepage)</td>
</tr>
<tr>
<td>10,461</td>
<td>has-publisher (dc:publisher)</td>
</tr>
<tr>
<td>7,308</td>
<td>has affiliation (foaf:workplaceHomepage)</td>
</tr>
<tr>
<td>5,850</td>
<td>in series</td>
</tr>
</tbody>
</table>


Furthermore, we evaluate range queries with a real dataset of approximately 11.2M triples. The DBLP dataset [1] is available in XML and contains a large number of bibliographic descriptions on major computer science journals and proceedings, more than half a million articles and several thousand links to home pages of computer scientists. For our evaluation, we use an RDF converted dataset from XML of the Proximity DBLP database, which is based on the DBLP dataset with additional preparation performed by the Knowledge Discovery Laboratory, University of Massachusetts Amherst. Specifically, the data in this dataset were derived from a snapshot of the bibliography in 2006. Tables 11 and 12 indicate important characteristics of the DBLP dataset that prelude our results. In addition, there
are 2,425,830 triples with `<rdf:type>` as their property, another 1,708,988 with `<http://xmlns.com/foaf/0.1/name>`, just 1,689,330 triples with a `<dc:creator>` predicate, etc. In fact, only seven distinct predicates appear in the outlandish percentage of 72% of our triples. Consequently, considering the immense arising skew, the hitherto approach of earlier efforts that hash each triple separately by its subject, its predicate and its object is absolutely unjustified. Hence, when the size of the overlay exceeds the number of keywords used in a dataset, especially when some of those terms are also popular, like `type` and `Article`, load imbalances have an immense impact on all aspects of performance and scalability. Therefore, we made some enhancements in the design of one of our competitors, RDFPeers [15], and benefited its overlay infrastructure to support multi-attribute range queries and not hash triples separately for each of their attributes. Otherwise, a single peer in RDFPeers would easily be responsible for at least half the dataset, a critical impediment to the functional operation of a peer-to-peer system.

We evaluate MIDAS-RDF with various types of queries that were used in our inference methods. More specifically, with `rangeXZ` we denote querysets consisted of range queries of the form `SELECT ?publication ?author WHERE {?publication <dc:creator> ?person}`, where only the predicate has a fixed value as in Algorithm 2. With `rangeZ` querysets of range queries of the form `SELECT ?author WHERE {<http://www.w3.org/TR/xquery> <ex:editor> ?author}`, where we want to retrieve triples of a specific subject and

<table>
<thead>
<tr>
<th>Frequency</th>
<th>Entities (in main classes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>560,792</td>
<td>Person (foaf:Person)</td>
</tr>
<tr>
<td>561,895</td>
<td>Articles in Proceedings</td>
</tr>
<tr>
<td>340,488</td>
<td>Journal Articles</td>
</tr>
<tr>
<td>10,610</td>
<td>Webpages of persons</td>
</tr>
<tr>
<td>9,027</td>
<td>Proceedings</td>
</tr>
<tr>
<td>2,530</td>
<td>Book Chapters</td>
</tr>
</tbody>
</table>

Table 12: Statistics summary of the DBLP dataset (Resources: 2,395,467, Literals: 3,064,704).

Figure 4: Latency of MIDAS-RDF, RDFPeers for `rangeXZ`, `rangeX`, `rangeZ` querysets on the DBLP dataset.

Figure 5: Latency and task-load for the computation of the deductive closure of synthetic graphs of variable depth for labeling and iterative based schemes.

Figure 6: Congestion - average peer task load.
predicate, for example in Algorithm 1. Last, with rangeX we denote range queries of the form \( \text{SELECT } ?\text{id} \text{ WHERE } (?\text{id} \text{ date } \text{ "2002-01-03"}), \) to retrieve all possible subjects that appear with a specific predicate and object (Alg. 4). In our queryset, the queries were originated from the DBLP dataset. They are triples selected at random and processed appropriately. Their querysize and selectivity is not fixed and is associated with the frequency of the used terms. Our querysets consist of approximately 40K range queries. Last, in Figures 4 and 5 we draw the mean values of important metrics, for the 20 times that we run our experiments.

6.2 Results

In this section we present our results that validate our analytical claims. Figure 4 presents query performance aspects for various types of range search, namely rangeX, rangeZ and rangeXZ. Latency for MIDAS-RDF is bounded by \( O(\log n) \) as expected. In terms of latency, the original version of RDFPeers performs equally good with our solution. Nonetheless, both data- and task-load fairness suffered severely in such a degree that commodity hardware would find it very difficult to cope with. On the other hand, our pure multi-dimensional scheme prevents similar phenomena as it takes load into consideration when assigning a zone to a new peer, and as a result, no peer is duly loaded. Regarding the altered version of RDFPeers that uses the protocol of MAAN to index triples, it is obvious that MIDAS-RDF outperforms it by more than an order of magnitude for all query types. Clearly, being strongly affected by query selectivity our competitor shows linear behavior as the overlay size increases, in contrast to MIDAS’ logarithmic performance. More specifically, latency on MAAN is dominated by the number of peers relevant to the query, due to the adopted approach to look up for a bound of the range first, and then sequentially traverse all relevant neighboring peers. This also becomes apparent for the various types of range search where performance deteriorates as selectivity diminishes from rangeX to rangeXZ.

Figure 5 presents query performance aspects for resolving the transitive closure of a synthetic RDF graph of variable depth. We compare MIDAS-RDF and its incorporated labeling scheme with the traditional iterative approach that has been widely accepted by similar efforts on the subject to date. Regarding latency, our approach performs logarithmically and clearly outperforms the iterative procedure. In particular, the usage of labeling schemes ameliorates performance significantly, as it renders obsolete the impact of graph’s depth parameter. Surprisingly enough, both labeling schemes we leverage, prefix and interval scheme, performed equally well with MIDAS.

When it comes to congestion, MIDAS-RDF outperforms the competition again. Apparently, resolving range queries is more efficient than the traditional iterative procedure of chasing pointers, some orders of magnitude analogous to the depth of the resolved graph. Also, note that this figure corresponds to the average peer message load when invoking \( n \) queries for retrieving the whole transitive hierarchy of an RDF graph that consists of 1M triples and variable depth.

Finally, Figures 6 and 7 illustrate in detail the progress of the transitive closure computation in MIDAS-RDF with time for the responsiveness and recall metrics.

7. CONCLUSIONS

This paper presented MIDAS-RDF, a novel distributed RDF repository based on a pure multi-dimensional indexing scheme for large-scale decentralized networks. MIDAS-RDF is able to process various pattern queries in hops logarithmic to the number of peers. Furthermore, using labeling schemes, MIDAS-RDF implements a forward-chaining inference method that outperforms known approaches that rely on iterative procedures. Last but not least, MIDAS-RDF supports a publish-subscribe model that enables peers to selectively subscribe to RDF content.

8. ACKNOWLEDGMENTS

The authors would like to thank Bettina Kemme and the anonymous referees for their valuable comments and suggestions to improve the quality of this paper.