Hannes Mühleisen
Freie Universität Berlin, GERMANY

Kathrin Dentler
Vrije Universiteit Amsterdam, THE NETHERLANDS

Large-Scale Storage and Reasoning for Semantic Data Using Swarms

Digital Object Identifier 10.1109/MCI.2012.2188586
Date of publication: 13 April 2012

1556-603X/12/$31.00©2012IEEE

Abstract–Scalable, adaptive and robust approaches to store and analyze the massive amounts of data expected from Semantic Web applications are needed to bring the Web of Data to its full potential. The solution at hand is to distribute both data and requests onto multiple computers. Apart from storage, the annotation of data with machine-processable semantics is essential for realizing the vision of the Semantic Web. Reasoning on web-scale data faces the same requirements as storage. Swarm-based approaches have been shown to produce near-optimal solutions for hard problems in a completely decentralized way. We propose a novel concept for reasoning within a fully distributed and self-organized storage system that is based on the collective behavior of swarm individuals and does not require any schema replication. We show the general feasibility and efficiency of our approach with a proof-of-concept experiment of storage and reasoning performance. Thereby, we positively answer the research question of whether swarm-based approaches are useful in creating a large-scale distributed storage and reasoning system.

I. Introduction

The success of the Semantic Web leads to ever-growing amounts of data that are being generated, interlinked and consumed. Handling this massive volume is a serious challenge, where scalable, adaptive and robust approaches are needed. Traditional approaches for handling data are often based on large dedicated computer systems which store all required data at one single location and handle all incoming requests from applications and their users. While this is a valid approach for limited amounts of data, it is no longer economically viable for web-scale data due to non-linear increases in hardware costs. Furthermore, robustness of a single system is always limited, making these single-node approaches less suitable for use in the envisioned Web of Data.

The apparent solution is to distribute both data and requests onto multiple computers. In this case, a method to create coherence between these computers is required, designed to make the distributed system appear like a single large unit to its users. Typically, these methods aim to minimize communication costs and to maximize the degree of coherence between nodes. Several methods have been researched and implemented, ranging from master/slave configurations to unstructured and structured Peer-to-Peer systems where all nodes share all responsibilities. Each method represents a trade-off between different dimensions, most commonly scalability, robustness and adaptivity [1]. Scalability is the system's ability to handle increasing amounts of data and requests, robustness is the ability to tolerate failure, and adaptivity is the ability to handle different data characteristics and various request patterns without the need for intervention. Sufficient performance along these three dimensions is required in a storage system for web-scale data.

To realize the vision of the Semantic Web, the annotation of data with machine-processable formal semantics is essential. From these schema annotations, reasoning engines make implicit information explicit, bridging the gap between data and knowledge. When this reasoning process is to be applied to web-scale data, the same requirements regarding scalability emerge, and distributing this task onto many computers is the only viable solution. However, fully distributed reasoning is scarce. Many previous approaches rely on central instances orchestrating the reasoning process, e.g. in [2], which is undesirable since these central nodes are not protected against failure per se, and their failure would inhibit the entire system to perform its reasoning tasks. Alternatively, schema information is replicated in all participating nodes, which is unfeasible in cases where the schema information is very large. Consistency cannot be ensured and any changes to the schema can result in substantial overhead.

Generally, manually configuring and operating large-scale distributed systems that potentially comprise of thousands of nodes is no longer feasible. *Self-organizing* distributed systems are able to operate autonomously [1] and are a promising solution to the challenge of handling distributed systems that provide large-scale storage and analysis for the web of data. The problem addressed by this paper is the design of a method for fully distributed storage and reasoning for Semantic Web data.

One approach to achieve self-organization is the collective behavior of individuals that cooperate in a swarm. The overall goals of the swarm, for example, to find food, are pursued through independent actions of the individuals based on indirect communication methods. From these local actions, a global, intelligent and coordinated behavior emerges [3]. Algorithms inspired by this behavior that aim to imitate the accomplishments of swarms with regard to their self-organization have successfully been applied to solve hard problems such as routing in computer networks [4]. As described above, the challenge of storing, reasoning on and retrieving large-scale semantic data in a distributed setting is a task that can only be sufficiently performed in a truly decentralized way. This makes swarm-based approaches interesting candidates for achieving

the desired self-organization. However, as every distributed system has to trade-off between different goals, these properties come at a cost. Swarm-based approaches trade deterministic guarantees to achieve their advantages. This might make these approaches unfit for use in database-like scenarios where the transactional paradigm has to hold. On the other hand, the extended work on NoSQL storage solutions, which also trade away guarantees in favor of scalability, indicates need for this type of storage [5]. Furthermore, handling web-scale data is a task whose dimensions are yet unclear and it can become necessary to make further compromises. Also, exhaustive results as expected from a classical database make little sense in a web scenario, where the information presented can only be a subset of the available data.

In this paper, we contribute our novel concept for both distributed storage and reasoning on Semantic Web data based on ant-inspired algorithms. We present how the ant's behavior may be adapted for distributed storage and reasoning. We strive to answer our research question of whether swarm-based approaches are useful in creating a large-scale distributed storage and reasoning system for semantic data. We present our concept of such a system in the subsequent Section II, and also contribute a "proof of concept" experiment of the storage and reasoning concepts on a real-world data set in Section III. We discuss related work in Section IV, and conclude this paper in Section V with a discussion of our results showing the feasibility of our approach.

II. Swarm-Based Semantic Storage and Reasoning

Distributed storage, retrieval and reasoning can all be reduced to locating the place where data is to be stored or retrieved to answer queries or to apply reasoning rules. To achieve scalability in these tasks, the location method needs to be as efficient as possible, while maintaining robustness and adaptivity. In the previous chapter, we have argued that self-organization is a method for sufficient performance in all dimensions. However, achieving self-organization with a number of pre-defined algorithms reacting on scenarios is not feasible, as the system will likely face situations not envisioned by its creators. Hence, built-in computational intelligence that is able to adapt itself to new situations is desirable [6].

From the large number of nature-inspired methods that are part of the research in Computational Intelligence, it has been shown that ant foraging is best suited to solve the location problem [7]. In this section, we first introduce basic Semantic Web concepts and technologies. We then describe the brood sorting and foraging methods found in ants and their application to distributed systems. We also show how they can be used to create a distributed storage system with reasoning capabilities. Both ant-based distributed storage and retrieval [8] as well as ant-based reasoning on Semantic Web data [9] have been proposed separately before. We describe how both swarm-based distributed storage and swarm-based reasoning can be combined and extended into a fully distributed and self-organized storage system for Semantic Web data with efficient and fully distributed reasoning. We show how both approaches can benefit from each other, forming a new system capable of performing these tasks with minimal overhead.

TABLE 1 RDFS entailment rules.

RULE	ANTECEDENTS	CONSEQUENT
rdfs2	p *rdfs:domain* c. s p o.	s *rdf:type* c.
rdfs3	p *rdfs:range* c. s p o.	o *rdf:type* c.

A. Semantic Web Concepts and Technologies

Semantic Web research has created the Resource Description Format (RDF) data model. An RDF graph is a directed graph, where the nodes are either URIs, literal values such as strings and integers, or graph-internal identifiers known as blank nodes. Directed and labeled arcs connect these nodes, URIs are also used for these labels. RDF provides a highly generic and flexible data model able to express many other more specific data structures, such as relational or object-oriented data. More formally, let $\mathcal{U}$ be a set of URIs, $\mathcal{L}$ a set of literals and $\mathcal{B}$ a set of blank node identifiers. Any element of the union of these sets $\mathcal{T} = \mathcal{U} \cup \mathcal{L} \cup \mathcal{B}$ is called a RDF *term*. A RDF *triple* is a triple (s,p,o), where $s \in \mathcal{U} \cup \mathcal{B}$, $p \in \mathcal{U}$ and $o \in \mathcal{U} \cup \mathcal{B} \cup \mathcal{L}$. An RDF *graph* is defined as a set of triples [10]. The query language SPARQL has been developed to express complex queries on RDF graphs. For convenient access to the data stored in such a graph, one may use a so-called triple pattern, which may contain variables instead of values. Variables are members of the set $\mathcal{V}$, which is disjoint from $\mathcal{T}$. A *triple pattern* is a member of the set $(\mathcal{T} \cup \mathcal{V}) \times (\mathcal{U} \cup \mathcal{V}) \times (\mathcal{T} \cup \mathcal{V})$. The declarative complex query language SPARQL is based on combining triple patterns and provides many additional features such as ordering and filtering [11].

RDF data (ABox) can be annotated by schema information in the RDF vocabulary description language RDF Schema or the web ontology language OWL (TBox), allowing an automated reasoner to make use of the semantics within an RDF graph. Both ABox and TBox are represented as RDF triples. In many cases (RDF Schema and parts of OWL 2), the semantics expressed by the schema can be calculated using a limited set of pre-defined reasoning or entailment rules. Each reasoning rule consists of two parts: the antecedent(s) and the consequent. The antecedents together specify a graph pattern that is to be matched to an RDF graph, and the consequent specifies how to generate a new triple, i.e. inference. RDF Schema (RDFS) entailment rules have one or two antecedents. If all variables contained in the antecedents are bound to values in the graph, the rule fires and the inference is created. The closure of an RDF graph under the RDFS semantics [10] can be derived by applying all RDFS entailment rules until no new triples are inferred (fixpoint iteration).

As an example, let us consider the RDFS entailment rule for *rdfs:domain* that is shown in Table 1. Whenever an RDF graph contains a triple that states that a given property p has the

rdfs:domain c, and makes use of this property in another triple, it can be inferred that the subject of the other triple is of *rdf:type* c. The entailment rule for *rdfs:range* works accordingly, but refers to the object of a triple that contains the property for which an *rdfs:range* is defined. Let us consider exemplary social network data that makes use of the FOAF vocabulary[1]. Figure 1 shows a limited extract of the corresponding RDF graph. The RDF triple on the bottom states that `Alice foaf:knows Bob`. The two schema triples state that the property `foaf:knows` has both `rdfs:domain` and `rdfs:range foaf:Person`. Based on all triples, it can be inferred that both Alice and Bob are of type `foaf:Person`.

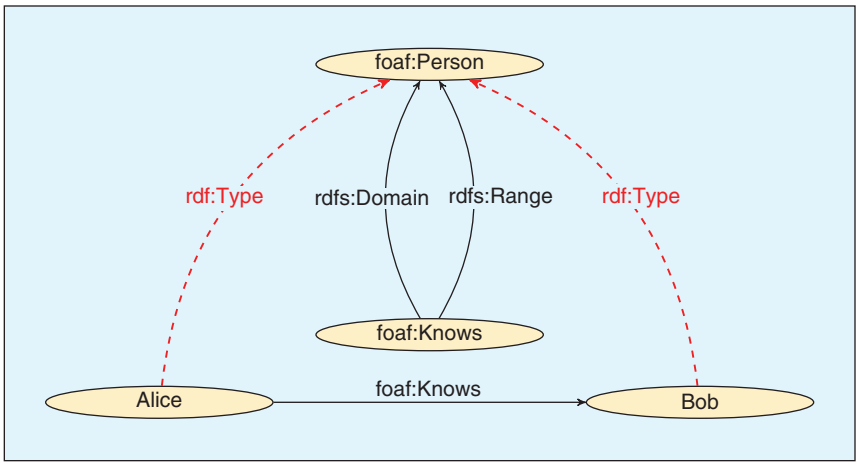

FIGURE 1 An exemplary RDF graph.

B. Swarm Intelligence for Distributed Systems

The basic notion of swarm algorithms is to employ a large number of simple, lightweight individuals. The overall goal of solving the specific problem never depends on single individuals, making those expendable and thus ensuring the robustness of the solution process. In general, each individual only has a *limited view* of its surroundings, e.g. only recognizing their immediate vicinity. Also, individuals only have *limited memory*, forcing them to make decisions based purely on locally available information with the help of *simple rules*. The decision process for a single individual can be described in very few simple behavioral rules. These three principles make swarm algorithms *scalable* with regard to the number of individuals, *robust* to the loss of swarm members and *adaptive* to ever-changing environments. While a single individual has no concept of how to solve the task which the entire swarm is facing, individuals are able to communicate indirectly by changing their environment. A global solution then emerges through the sum of all single independent actions of the individuals.

From a multi-agent perspective, ants are very simple agents which only communicate indirectly via the landscape, which serves as a collective memory. The combination of the independent local actions of the individuals represents a global optimization mechanism based on positive feedback.

The first method that is useful to solve the location problem in a distributed system is the foraging method by which ants search for food in colonies consisting of thousands of individuals. A subgroup of the ants is assigned to search for food and bring it back to the nest. They do so by leaving their nest, randomly changing directions at first. As soon as they encounter food, they carry it back to the nest. On the way back, they leave a chemical pheromone trail behind. Now, other ants who are also searching for food can smell this trail, which leads them to follow it with increasing probability for increasing pheromone intensity. This way, the pheromone intensity of paths to rich food sources is being reinforced, as more and more ants use this path. Ants leaving the nest now no longer have to wander around randomly, but can choose between established paths. Pheromones evaporate over time, so that paths to depleted food sources will disappear [12].

The second ant-inspired method that can be useful for data organization is the method by which ants sort their brood. To optimize care for their brood, ants cluster larvae according to their development stage. To solve this clustering problem, ants developed a fully decentralized method where the ants inspect the larvae nearby and then pick up the most dissimilar one. If they are carrying around a larva, they are inclined to drop it where similar larvae are placed.

The advantage of both methods from a distributed system perspective is that they do not require any shared global data structure, which would have to be reliably maintained at considerable cost. Rather, these methods can be applied in a fully distributed system and still maintain a high degree of efficiency [4]. The tradeoff for this high performance with low requirements is a small degree of failure probability, for which a recovery method has to be designed.

C. Storage and Retrieval

Typically, a distributed storage system consists of a large number of fully independent computers, which are connected by a network. The general goal for this set of computers (nodes) is to provide one single storage service, with data being distributed across all nodes. Since a central gateway to the service provided by the computer network would be a single point of failure, each node should be able to serve requests for all data that is being handled within the system. Storing and retrieving data items is now a problem of locating the subset of nodes within the network which are responsible to store the information. No single node can have all knowledge that is required, since this would be both a bottleneck and Achilles' heel for the system. The problem thus has to be solved through

[1]FOAF: http://xmlns.com/foaf/spec/

cooperation of multiple nodes based on partial information that is locally available. The smaller the subset of involved nodes, the higher the scalability, and typically a logarithmic proportion of the total number of nodes is considered sufficient. We have already proposed the application of ant foraging to solve this problem [8].

The swarm-inspired algorithms are adapted to our distributed system as follows: The internal storage operations are considered the swarm's individuals moving around on a virtual landscape of nodes connected using network technology. Every node is connected to a limited number of other nodes that are its so-called neighbors. The average number of neighbors is equivalent to the degree of the overlay network topology. The data to be stored inside this network is modeled as the ant's food or larvae, respectively. From a Peer-to-Peer perspective, this approach represents a compromise between unstructured and structured networks. Structured, because the pheromone trails represent a shared data structure dramatically improving routing efficiency, and unstructured, because a node's position in the network is dynamic and its routing decisions are not determined by a global law, but rather on a best-effort local heuristic exploiting purely local information.

The ants' brood sorting method is used for write operations. We calculate a so-called numerical routing key for each data item based on a similarity measure. In the general case, the similarity measure is a hash function on the literal key value, e.g. SHA1. Similarity between two items is their numerical distance of the routing key values. Depending on the notion of the distance to be used for the stored data, other similarity measures such as numeric similarity can be used [13].

The routing key then enables the individuals to find an area of the storage network where similar items are stored and where new data items are thus placed. In the case of RDF, graphs to be stored are deconstructed into RDF triples and each triple becomes part of three separate write operations, each time with another triple component (subject, predicate, object) as a routing key. The write operations then move from node to node until they find a number of triples sufficiently similar to the triple that is to be stored, and then they will store it. This leads to clusters of triples that use the same or similar routing keys being placed on the same or neighboring nodes, generating a global degree of organization in the storage network. The details of this process are described in [8]. Since routing is not based on a global law but rather on individual decisions at each storage node, uneven data or request load of single nodes the storage network is unproblematic. Excess data can be moved off to neighboring nodes, and corresponding requests will first be forwarded, with the routing heuristic updating itself to reflect the new location soon [14].

If a node receives a request for information, it creates an internal read operation which is also able to move from node to node similar to the foraging method, thereby regarding RDF triples as food. Triples can be searched for by specifying both fixed values and variables for any triple entry. Thus, any basic graph pattern can be evaluated as a search pattern that is to be matched against the triples. Again using the similarity measure, the read operation is able to find the part of the network where the cluster containing the particular data item is stored, thereby exploiting the locality created by the brood sorting. Successful operations return their result to the waiting application at the node where the query originated and trace back the path they have taken. They use the calculated routing key to intensify virtual pheromones maintained for each connection to another node on the path taken. These pheromones are distinct for each key that has been used to store a data item, the added intensity is dependent on the size of the result set and the path length. While the operation has not yet arrived inside the cluster where the searched triple is located, the routing only has to find this cluster. Hence, pheromone values can also be compressed through aggregation into ranges, effectively limiting the space needed to maintain the pheromones.

As in nature, subsequent operations can read these pheromones and calculate the likelihood of finding results by traversing that particular connection and pheromone values decrease over time to simulate evaporation with a configurable decay rate. Should pheromone values be absent or ambiguous, a random node is chosen as a next hop, with the randomness decreasing as the pheromones get more intense.

The repeated process of requesting different data items from different locations will create a multi-layered network of pheromone paths leading from the nodes where requests were received to the nodes where data was received. The self-optimizing property of the swarm method used will lead to a near-optimal path through the network, as shorter paths are assigned stronger pheromone intensities. The optimal length of the paths is dependent on the properties of the overlay network, which is dependent on the amount of neighbor nodes each node has. Hence, the retrieval costs for often-requested large clusters is close to the shortest path between the requesting node and the node storing the data item. For unpopular and small clusters, the retrieval process can degrade to a random walk, which can incur costs linear to the amount of nodes in the network once. Since it is also possible that a retrieval operation starts a circular movement pattern, its number of steps between nodes is limited by configuration. Should the operation reach the maximum number of steps allowed, it fails and reports back to the node it originated from. It is then able to restart the retrieval operation.

The retrieval process is shown in Figure 2. Here, a request for a triple with the key #B is received at node S2. From the pheromones present for this key, the likelihood for finding matching triples on the connected nodes S1, S5 and S6 can be calculated. According to this probability distribution, a weighted random routing decision is taken, in this case most likely routing the request to node S1, from where it is again likely that the operation will find the searched data item on node S3. However, should this not be the case, every other node is also able to calculate these probabilities, thereby achieving an increasing probability that the operation will find the data item with further hops.

Evaluating complex queries inside this system is still another area of ongoing work [15], but in a first step we have implemented the storage interface used by a general-purpose SPARQL processing engine. The static optimizations used by this engine create a sequence of retrieval operations, which are then executed as series of single retrieval operations for each triple pattern in the query. The results of all retrieval operations are collected and added to a temporary graph, on which the complex query can then be evaluated. While effective, efficiency of this approach can be impaired by large intermediate result sets.

D. Reasoning

Two approaches to infer implicit knowledge are forward and backward chaining of reasoning rules. The idea behind forward chaining is to derive and optionally materialize all possible inferences based on input data, typically when new statements are inserted into a triple store. In contrast, backward chaining of reasoning rules is usually performed during query processing, applying rules which lead to the results that are being queried for in reverse order. Backward chaining thus requires less storage at the cost of longer query processing times. In our usage scenario, we have large amounts of space available to store inferred information and aim at fast query processing. Thus, we focus on forward chaining only.

We previously presented a distributed, forward chaining reasoning method based on swarm intelligence [9]. There, individuals of a self-organizing swarm "walk" on the triples of an RDF graph, aiming to instantiate pattern-based inference rules. To apply this approach to the self-organized semantic storage service, we adapt the members of the swarm so that they no longer employ pheromones to traverse the RDF graph structure, but to find nodes with triples that they can apply their inference rules on. Our storage layer does not differentiate between data (ABox) and schema information (TBox), which are both encoded as RDF triples. Hence, we assume all schema information to be part of the entire set of triples that is stored in the storage network. The forward chaining of reasoning rules can be applied by swarm individuals, assigning each reasoning rule to a number of individuals, who subsequently traverse the network trying to find matches for the antecedents and creating inferences whenever the rule fires. Contrary to the previous approach, we are able to re-use the pheromone trails left behind by the storage operations to efficiently locate the triples required to calculate inferences.

By relying on a storage layer based on a similar concept, the proposed reasoning method requires no additional distributed data structures in the network nor on the nodes and still is able to express increased efficiency. Regarding the expressivity of this approach, methods for sound and complete distributed resolution on the Description Logic $\mathcal{ALC}$ have been proposed [16]. Supporting the same expressiveness in a swarm-based system has also been shown to be feasible together with a formal discussion of the theoretical correctness of the approach [17].

In our system, each node periodically scans the locally stored triples for values that are contained in the triple patterns of the antecedents of the pre-configured rules. Each match becomes a new reasoning operation, which is initialized with the now partially bound triple pattern or basic graph pattern. The reasoning operation now tries to find triples that complete the remaining triple patterns that belong to the basic graph pattern by moving through the network using the previously bound values as routing keys. To find potential matches, the pheromone paths are exploited to efficiently route them to their next destination, even if reaching the destination node requires several hops. Once all variables are bound, the rule fires and the inferred triples are written to the storage network using the write operation. This has the advantage that triples which are already present in the store are not added again, so that duplicates resulting from the reasoning process, for example due to several rules that lead to the same inference, are not added multiple times.

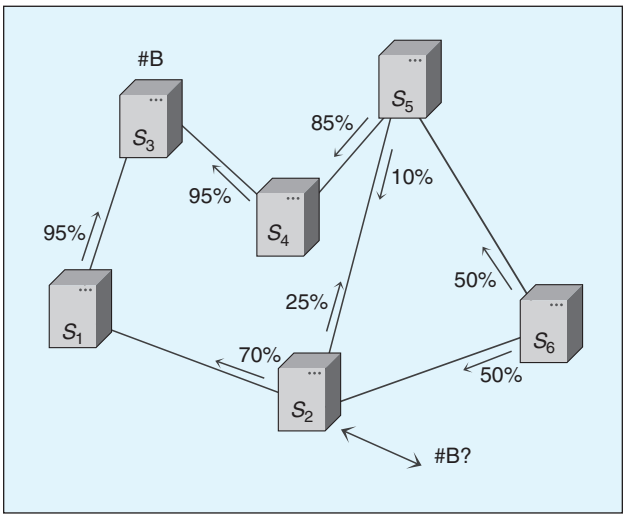

FIGURE 2 Network structure with routing probabilities.

We will demonstrate how the reasoning process is executed using the RDF Schema entailment rules [10]. As RDFS entailment rules with only one antecedent are trivial, we will focus only on rules with two antecedents. Triples matching the antecedents could be stored on different nodes, which requires data exchange between those nodes in order to fire the reasoning rule. This subset of the RDF Schema specification contains `rdfs:domain` and `rdfs:range` entailments, the transitive closure and the implications of `rdfs:subPropertyOf` and `rdfs:subClassOf`. All considered rules contain at least one schema triple, that is antecedents that contain an element from the RDFS namespace. Table 2 shows the different phases of our approach for these inference rules. First, the node-local store is searched in the **init** phase for triples that can be matched to antecedents that are schema triples, and new reasoning operations are generated and initialized based on the found triples. In the **move** phase, these reasoning operations now search for matching triples, first locally and then traversing the storage network. For our employed RDFS rules, one (rdfs2, rdfs3, rdfs7 and rdfs9) or two elements (rdfs5 and rdfs11) need

TABLE 2 Application of RDFS entailment rules.

RULE	INIT	MOVE	INFERENCE
rdfs2	**p** *rdfs:domain* c.	S **p** O.	s *rdf:type* c.
rdfs3	**p** *rdfs:range* c.	S **p** O.	o *rdf:type* c.
rdfs5	P_1 ***rdfs:subPropertyOf*** $\mathbf{P_2}$	$\mathbf{P_2}$ ***rdfs:subPropertyOf*** p_3.	p_1 *rdfs:subPropertyOf* p_3.
rdfs7	$\mathbf{P_1}$ *rdfs:subPropertyOf* p_2.	S $\mathbf{P_1}$ O.	s p_2 o.
rdfs9	$\mathbf{c_1}$ *rdfs:subClassOf* c_2.	s *rdf:type* $\mathbf{c_1}$.	s *rdf:type* c_2.
rdfs11	c_1 ***rdfs:subClassOf*** $\mathbf{c_2}$.	$\mathbf{c_2}$ ***rdfs:subClassOf*** c_3.	c_1 *rdfs:subClassOf* c_3.

to match; they are printed in bold and underlined in the table. If a reasoning operation encounters a match, it creates an **inference.**

Regarding our foaf example from Section II-A, in the **init** phase, two reasoning operations are generated: one for the domain axiom and one for the range axiom. In the **move** phase, these reasoning operations search for triples that contain the necessary elements for their inference rules to be applied. Both the `domain` and the `range` reasoning operations now search for occurrences of the predicate `foaf:knows`. Let us assume that data using this predicate does not occur at the local node, so that the reasoning operations migrate to other nodes searching for it. Whenever a match is found, the reasoning operation creates a newly derived triple. For example, the reasoning operation responsible for the domain axiom finds a triple `Alice foaf:knows Bob` and **infers** that the subject of this triple, i.e. `Alice` is of type `foaf:Person`.

ALGORITHM 1 Reasoning Operation—Initialization Phase.

Require: Set of reasoning rules s_r

1: **for all** $r \in s_r$ **do**
2: $p \leftarrow$ *init Pattern*(*antecedent*(r))
3: $s_t \leftarrow$ *localRead*(p)
4: **for all** $t \in s_t$ **do**
5: $p_b \leftarrow$ *bind*(p,t)
6: $r_b \leftarrow r \setminus p \cup p_b$
7: *createInferencingAnt*(r_b)
8: **end for**
9: **end for**

ALGORITHM 2 Reasoning Operation—Move Phase.

Require: Partially bound reasoning rule r_b
Require: Hop limit h_{max}
1: $h \leftarrow 0$
2: $a \leftarrow$ *antecedent*(r_b)
3: $p \leftarrow$ *movePattern*(a)
4: **if** *allBound*(a) **then**
5: **return** *bind*(*consequent*(r_b),a)
6: **end if**
7: **while** $h < h_{max}$ **do**
8: $s_t \leftarrow$ *localRead*(p)
9: **for all** $t \in s_t$ **do**
10: $p_b \leftarrow$ *bind*(p,t)
11: $r_b \leftarrow r \setminus p \cup p_b$
12: *createInferencingAnt*(r_b)
13: **end for**
14: *move*(*routingKey*(p))
15: $h \leftarrow h + 1$
16: **end while**

A more formal definition of the reasoning process is given as pseudocode. The process consists of two phases: First, the initialization phase scans the local storage on every node for ABox (schema) triples, and instantiates reasoning operations that move through the network. This process is given in Algorithm 1. For a set of reasoning rules, the local storage is scanned for triples matching the init pattern of the reasoning rule (Line 3), which is part of the configured rules given in Table 2. For each matching triple, the found values are bound to the pattern, and a new reasoning ant is spawned with this pattern and sent on its way (Line 7).

The process for handling these reasoning ants on every node they visit is given in Algorithm 2. Here, the local storage is checked for triples matching the move pattern (Line 8). For every match on the current node, the match is bound to the antecedent pattern, and a child reasoning ant is spawned (Line 12). If no matches are found, the reasoning ant is routed to the node storing matching data using a routing key from the move pattern. Both move pattern and routing key are also pre-defined in the current reasoning rule set. If the antecedent pattern is fully bound, the resulting triples can be created using the bound values from the antecedent and the static values from the consequent (Line 5). The new inferred triple can now be written using the write operation described above.

Our approach has several advantages: 1) it is fully decentralized, since every node that happens to store a part of any compatible schema will create the corresponding reasoning operations, which do not need to be controlled in any way; 2) it does not require the replication of schema information; 3) it shows "anytime" behavior, generating sound inferences that can be queried for during the reasoning process, with the degree of completeness increasing over time. New triples can be added to the store at any time, leading to new inferences. When triples are deleted, we currently cannot trace and delete the inferences

that they caused, thus only monotonic logics are supported. This problem could be solved by adding a time stamp to each new inference, so that triples that are not re-inferred can be deleted after a while.

The advantages of our approach are gained by relinquishing completeness guarantees. By relying on the pheromone trails also used by the storage operations, the reasoning operation cannot guarantee that a part of a basic graph pattern that may be present somewhere in the network is actually found. However, at web-scale, incomplete reasoning methods have been found to be advantageous due to the gained robustness and scalability, and partial reasoning results are often useful as well [18, 19]. Furthermore, triples that are frequently requested have stronger pheromone paths leading to them, enabling the reasoning operation to find these triples more reliably. This leads to inferences on heavily-used data being calculated more quickly than others, while still maintaining the full adaptivity and robustness of our approach.

E. Stochastic Scalability Analysis

To determine the theoretical performance of routing heuristics, we will now perform a stochastic analysis of our operations. To this matter, we describe the average case performance of the retrieval operation. Consistent with distributed systems research, the unit of cost for this analysis will be hops, that is the amount of transitions of the operation between nodes [20], [21]. Typically, a logarithmic relationship between the amount of nodes in the system and the average hops required to find a data item is required for scalability.

Since a request can be started at every node and results can be on any node in the network, the cost for any retrieval operation is at least the distance between the nodes in the network. In the average case, this distance is the average path length in the network. Disregarding the possibility of the network graph having small-world or scale-free properties, we assume the average path length in random networks as our average distance from start to target node. The average path length in a random network l_{ER} (and also the average distance between nodes) is calculated as follows [22]:

$$l_{ER}(N, \langle k \rangle) = \frac{\ln N - \gamma}{\ln \langle k \rangle} + \frac{1}{2}$$

with N being the number of nodes, γ being the Euler-Mascheroni constant (≈ 0.5772) and $\langle k \rangle$ being the average connectivity in the network (equivalent to the average number of neighbor nodes).

Since our routing method is based on positive feedback, we can assume p_f to be in the range $[0, 0.5]$. For every step on the way from the origin to the destination node, three outcomes of the heuristic-supported routing process are possible: Positive, where the operation got one step closer to its destination, Neutral, where the amount of steps remaining is unchanged, and Negative, where the operation now is one step further away from the destination. Since network connections are defined to be bidirectional, a step in the wrong direction can add at most one additional step to the remaining path length. However, the distribution between neutral and negative outcome is unknown, we therefore introduce a second parameter, p_n. The probabilities for each case are thus as follows:

- $p(positive) = 1 - p_f$
- $p(neutral) = p_f * (1 - p_n)$
- $p(negative) = p_f * p_n$

For a single step in the network, the total impact i on the remaining path length is thus calculated as $i = -(1 - p_f) + (p_f * p_n)$. If the assumption of p_f being at most 0.5 holds, we can see that p_n has to be 1 in order for the improvement i to evaluate to 0. However, it is unlikely that every mistake adds another step to the operation's path, and hence we can safely assume p_n being smaller than 1. If this assumption holds, the improvement i is always negative. Consequently, every routing operation will bring the operation closer to its destination.

The expected value for the average hop count to retrieve an arbitrary element from the network is then the fraction of the average path length by the absolute value of the expected reduction of the remaining path length per hop.

$$\text{hops}(N, \langle k \rangle, p_f, p_n) = \left\lceil \frac{l_{ER}(N, \langle k \rangle)}{|-(1 - p_f) + (p_f * p_n)|} \right\rceil.$$

For example, in a network of 10,000 nodes with an average amount of 10 neighbors, the average path length inside the network is 10. If we assume a high routing error probability of 40%, and a realistic 50/50 distribution between neutral and negative cases, we expect on average 15 hops to reach the node where matching data is stored, or nine more than required from the network structure.

To come back to our stochastic analysis of the average amount of hops required to route any request from the node it was created on to the node storing matching data, we have shown the average amount of hops required to perform this task within our swarm-based system. As we have seen, the average amount of hops is dominated by the average path length, since the probabilities are independent of the network size. Hence, the overhead produced by our swarm-based approach is constant with regard to the network size, and thus our approach can be considered scalable.

F. Network Management

To create the overlay network, we propose a distributed network bootstrap protocol: New nodes are given the address of a "bootstrap" node that is already part of the network. The new node can now retrieve a list of neighbor nodes from the bootstrap node and request its addition to this list. This request is granted if the bootstrap node has not reached its neighbor upper limit as per its configuration yet. This process is then recursively repeated on the newly known nodes until the number of neighbors on the new node has reached the neighbor lower limit, also defined in the node configuration. If the number of bootstrap nodes is limited, this algorithm uses preferential

attachment to create a power-law network structure [23]. Nodes maintain connections to their neighbor nodes, and nodes not responding are removed. If the number of neighbors should fall below the lower limit, the bootstrap process is resumed.

G. Summary

We have designed the operations within the proposed distributed system according to behavior found in ants for foraging and brood sorting. These behavioral descriptions adhere to the general swarm properties with landscape-coordinated actions of large numbers of individuals with limited view. Through the separation of the virtual landscape onto many nodes with a individually managed limited data structure, we have removed global state from the network, which is a major hindrance for scalability, while maintaining statistical efficiency. By design, storage, retrieval and reasoning operations as described above also do not require global state, and every information they require can be calculated on the node the operation is currently executed on. Therefore, the design dimensions for distribution – scalability, robustness and adaptivity are met for storage, retrieval and reasoning with data in the RDF model.

Scalability to the amount of data that can be stored in the network only depends on the sum of storage available on the individual nodes. Should this space become the limiting factor, new nodes can be added using our bootstrap protocol without affecting the entire network. As soon as the new node finishes the bootstrap protocol, the system-inherent randomness will lead to operations being routed to this node. As soon as data is placed on the new node, subsequent retrieval operations will create the pheromone paths leading to this data. This makes this new data efficiently available for retrieval as well as for reasoning operations.

Should individual nodes fail, new operations cannot be routed to these nodes anymore, and operations will have to be routed to the neighbor node with the second-strongest pheromone path from the neighbor list. Again, this does not affect the other nodes in the network, and after a limited time, operations would have created new paths "around" the unresponsive node, expressing the sought-after robustness of the proposed system. To keep the data formerly stored on this node available, a purely local replication scheme may be used.

Finally, adaptivity to skewed data is also possible: For example, if the distribution of routing keys is very uneven in the stored data, the data for this key may likely exceed the storage capacity of a single node. In this case, this node can individually decide to move a portion of the data to neighboring nodes. Pheromone paths will adapt to this new distribution in this area of the network, again keeping all reconfiguration in a very limited area of the network. Furthermore, pheromone paths will become stronger for much sought-after data. Operations requesting this data will exhibit a higher efficiency than operations for less popular data, leading to the designed statistical efficiency of the system.

The trade-off for these characteristics is the potential for failure, which can lead to failed retrieval operations, misplaced data items, and missed inferences. However, retrieval operations can always be restarted until the data is found and misplaced data items can be moved as shown through internal cleanup operations. Reasoning events are repeated periodically, eventually finding most possible inferences. Added operations will create additional load, making the performance of the entire system dependent on its capability to handle large amounts of operations. Should this become a bottleneck, new nodes could be added.

In spite of the conceptional fitness, the emergence of coordinated collective efficient behavior as aspired by their application cannot be proven but only shown in experiments. We present such experiments in the following section.

III. Experimental Results

To test our concept in a preliminary experiment of large-scale storage operations and reasoning in a distributed setting based on swarm intelligence, we have chosen a series of black-box tests in an experimental setup closely resembling the environment where the designed system is to operate. This method was chosen due to the properties of the employed ant algorithms with their inherent randomness, which makes results from simulations difficult to transfer. We have thus implemented our concept as a stand-alone software program, which was then run on a number of independent computing nodes rented from Amazon's Elastic Computing Cloud (EC2). Using our bootstrap protocol, the nodes created a network of connections among them, ensuring that each node is able to reach any other node in the network over at least one path.

For test data, we have used a subset[2] of a crawl from the Web of Data created for the VisiNav system [24] containing a large number of resources annotated using the Friend of a Friend (FOAF) vocabulary[3]. This subset containing ca. 75K triples was written using the storage operation described in the above section, effectively distributing as well as clustering the data over the nodes participating in the storage network. The FOAF vocabulary encoded in RDFS was also written to the network, so that the reasoning ants could be created by the system. This data set was chosen because it contains several characteristics we have identified to be problematic for other—more formal—approaches: First, it contains live web data, which are unchecked and messy, and which can bring logic-grounded reasoners without special optimizations to their limits [25]. Second, since the data is collected from several sources, the distribution of terms is unknown a-priori and potentially skewed [26], making it impossible to configure a conventional large-scale storage system beforehand. Third, a large amount of instance data is using a very small number of classes as defined by the schema, challenging approaches which move all potential matches for inference rules to the node storing the rule.

The relatively small size of the data set was deliberate to allow a large number of repetitions of the experiment, as the swarm algorithms always exhibit a degree of randomness, and multiple repetitions have to be performed in order to create a statistically significant result. As shown in the previous section,

[2]Test data set available at: http://beast-reasoning.net/a.nt
[3]FOAF vocabulary specification: http://xmlns.com/foaf/spec

the potential amount of data that can be handled is directly dependent on the number and storage capacity of the nodes and the throughput of the network connections between them. Hence, a larger data set would not yield additional insight into the system's behavior.

A. Storage and Retrieval

As described above, the main challenge for storage and retrieval operations in a distributed system is to find the node where a data item should be placed or searched for while involving as few nodes as possible. Rather than focusing on evaluating complex queries as described, we issued a retrieval request for a single arbitrary but fixed triple already stored inside the network to every node participating in the storage network. For each retrieval operation, the nodes taking part in the location of the triple were recorded. From the total number of nodes, the number of nodes not able to produce the triple were used to calculate a response success percentage. We have repeated this experiment over network sizes ranging from 20 to 150 nodes.

The results for one of these experiments are given in Figure 3. For the different network sizes and all queried nodes, the average and median number of hops required to find a single triple are plotted. These average values clearly show the average number of nodes to be far less than the number of nodes in the network. The variation in the values between network sizes are attributed to the employed randomness. Also, the response success percentage is 100% almost every time, confirming our expectations for the storage performance of our swarm-based approach. However, for this experiment, the correlation between network size and hops required was not linear. We suspect this to be due to randomness inherent in the swarm algorithms.

Hence, we have repeated the experiments shown in Figure 3 ten times in an effort to sufficiently remove the effects of randomness. Figure 4 shows these results. From the fitted curve, we can observe a linear increase at worst in the number of hops required to retrieve a triple over the network sizes. To determine the overhead created by the swarm-based approach, we have also statically analyzed the network structure created by the bootstrap algorithm between the nodes. We have found that the average path length inside these networks ranged between 1.5 for 20 nodes and 2.5 for 150 nodes. The maximum path length ranged from 3 for 20 nodes to 4 for 150 nodes. An optimal algorithm solving the location problem, for example, enjoying a global view on the network, would have been able to retrieve the data items using these optimal values for the number of hops. Hence, we are able to determine the overhead created using our approach in this experiment to be on average approximately two times the hops required by the "perfect" algorithm. Furthermore, previous research on foraging-based distributed systems has shown that the hop count required scales logarithmically with the network size in simulations [4]. Even though the number of nodes in this experiment was insufficient to prove this behavior for our approach in our limited testbed, a general trend towards logarithmic behavior is visible.

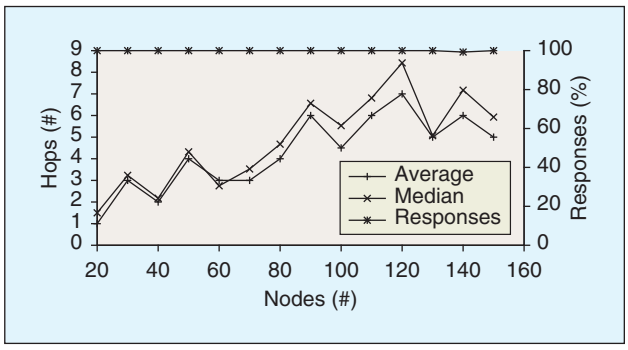

FIGURE 3 Hops required to find a triple for different network sizes.

B. Reasoning

The swarm-based method to perform basic RDFS reasoning as presented in the previous section was evaluated using a different method. From our test data set and the corresponding schema, we have calculated the RDFS closure using a conventional reasoner, which was able to calculate the closure after some problematic statements have been removed. The closure contained approximately the same number of triples as the original data set. Over 87% of the generated statements connected two resources with the `rdf:type` property, since those are most commonly generated according to the employed RDFS inference rules. In our data set, the number of `rdf:type` statements went from 10,173 to 76,734 statements after inferencing. We have focused on `rdf:type` statements in our distributed case, since discerning between static and inferred triples is non-trivial, as they are located in the same storage layer. The comparison of the number of these statements between the data generated by the reasoner and the data generated by the distributed process will yield the degree of completeness achieved by our reasoning process. To this end, each node was extended with a method to allow it to be queried for the number of those statements.

For a test protocol, we completed the process of writing the data set and the schema to the network, which we have limited to 50 nodes for the reasoning experiments. The previous experiment has shown that the number of nodes in the network is not the decisive factor in the system's performance. Then, the

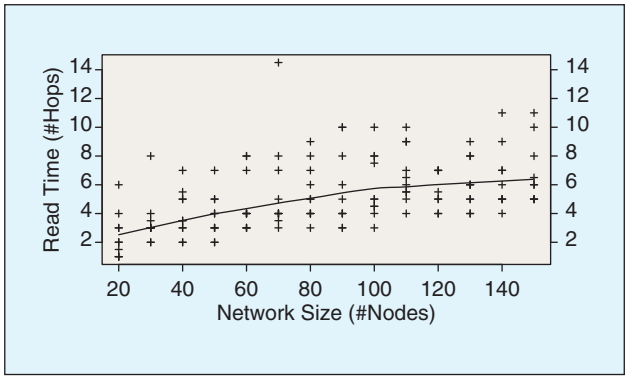

FIGURE 4 Hops required to find a triple for different network sizes—ten test runs.

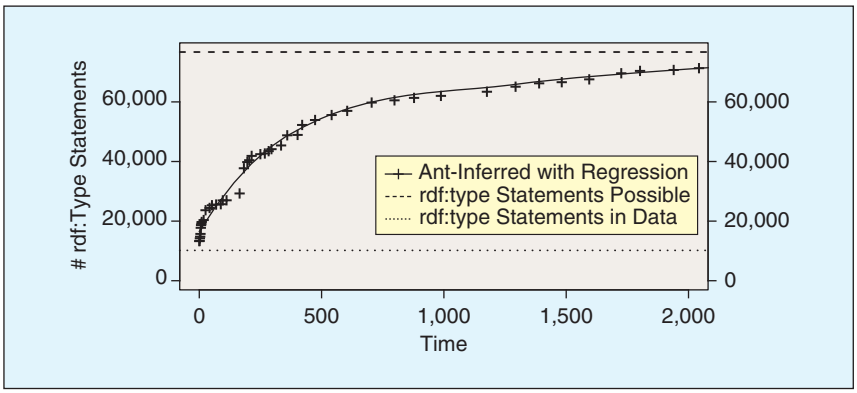

FIGURE 5 Inferred statements over time on 50 nodes.

reasoning process as described was started on each node, generating the corresponding reasoning operations. During this phase, we periodically measured the total number of `rdf:type` statements. Even though the basic scale of the experiment is time, the actual time values are not relevant, since they are highly dependent on the implementation of the system.

Figure 5 plots the results of a single test run, with two lines marking the number of measured statements already in the data set as a baseline, as well as the number of measured statements in the previously calculated full closure of the data set. The plot commences directly after the reasoning operations have been started, showing a discrete measurement along a time scale along with a regression line. The shape of the graph shows the saturation process typical for swarm-based reasoning [9]. We have repeated this experiment several times to remove singular influences by the system-inherent randomness. An aggregation of all experiments is plotted in Figure 6, converging on the same result as shown before, with the measurements exceeding the theoretical limit of possible inferences due to temporarily misplaced duplicates.

IV. Related Work

Our survey of related work is focused on distributed systems that either provide storage and retrieval on RDF data with reasoning capabilities or systems that support distributed reasoning as their only service.

Battré et al. [27] describe a method to perform distributed RDFS forward-chaining reasoning in their BabelPeers system organized using a distributed hash table (DHT). Through the properties of the distributed hash table along with their distribution scheme, they show how all triples required to evaluate the preconditions of the reasoning rules have to be stored together at one of the nodes. Thus, they are able to calculate possible inferences in a completely de-centralized process and then use the same process to materialize triples into the store. To solve the load balancing problem inherent in term-based partitioning, they introduce an overlay tree structure able to split the data with colliding hash values onto several nodes. However, this forces them to replicate schema information across nodes.

Fang et al. [28] presented an approach for distributed reasoning, where they first perform reasoning on the schema (TBox) using a Description Logic (DL) reasoner and then use the schema closure to create reasoning rules that are applied to the instances stored in a DHT (ABox). The schema is assumed to be present on all nodes, allowing any stand-alone reasoner to create the schema closure locally. To apply the reasoning rules to the instances, a "prefetch" operation retrieves the instance data that potentially match the reasoning rules to the local machine. After calculating the inferences, the results are distributed to other nodes, where they trigger further reasoning and reach the closure after multiple iterations of the process. The main issues with this approach are, again, the need for complete schema information on all nodes as well as the prefetch of data. For example, consider all instances in the data using a single RDF class. If this class is also defined to be the sub-class of another class, every node needs to load all the instances of the first class from all other nodes in order to evaluate the rule.

Kaoudi et al. [29] study the trade-offs between distributed forward-chaining and backward-chaining on RDF data on top of a DHT storage network and present their own algorithm on distributed backward-chaining with recursive lookup operations based on values from the query and the set of RDFS reasoning rules matching the query. For example, if the type of a resource is queried, they traverse the subclass hierarchy of the schema potentially stored at various parts in the network, ultimately determining all classes a resource is an instance of. They also argue against the scalability of distributed

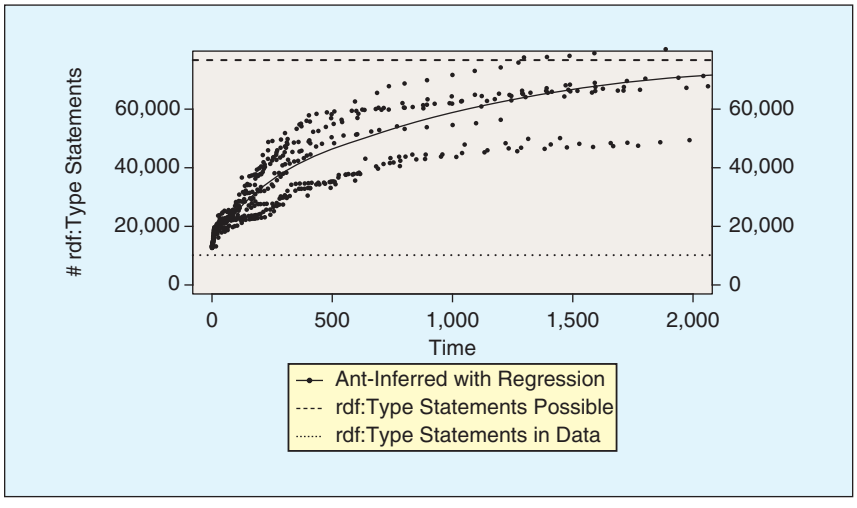

FIGURE 6 Inferred statements—repeated experiments.

forward-chaining in DHTs based on experimental results with a very small data set.

Marvin [30] is a platform for distributed reasoning on a network of loosely coupled nodes. The authors present a divide-conquer-swap strategy and show that the model converges towards completeness. Their routing strategy combines data clustering with randomly exchanging both schema and data triples. On each node, an off-the-shelf reasoner computes the closure. To handle the problem of inferred duplicates which cost memory and bandwidth, the authors propose a "one exit-door" policy, where the responsibility to detect each triple's uniqueness is assigned to a single node. This node uses a Bloom filter to detect previously hosted triples, marks the first occurrence of a triple as the master copy and removes all subsequent copies. For large numbers of nodes, a sub-exit door policy is introduced, where some nodes explicitly route some triples to an exit door. This incurs additional bandwidth costs to send triples to these exit doors.

Kotoulas et al. [26] show that widely employed term-based partitioning (such as in [27], [28] and [29]) limits scalability due to load-balancing problems. They propose a self-organized method to distribute data by letting it semi-randomly flow in the network, which allows clustered neighborhoods to emerge, and implemented it on top of Marvin. Both schema and data triples are moving in the network. A drawback of both Marvin-based approaches is that they solely rely on weighted randomness to ensure that data and schema triples come together at some point. As the number of nodes increases, we expect this to become increasingly less likely.

Urbani et al. [31] propose a scalable and distributed method to compute the RDFS closure of up to 865M triples based on MapReduce. One of the crucial optimizations is to load schema triples into the main memory of all the nodes, as the number of schema triples is usually significantly smaller than the number of data triples, and RDFS rules with two antecedents include at least one schema triple. We deliberately abandon this option, as we aim for an approach that is scalable and adaptive for all kinds of distributions among schema and data triples. Furthermore, replicating the schema information is no longer applicable when dealing with rules that contain two or more data triples as antecedents. In subsequent work[32], the approach was extended to the OWL Horst [33] semantics, able to deal both with required joins between multiple instance triples and multiple required joins per rule. The authors demonstrate the scalability of their approach by calculating the closure of 100 billion triples.

Salvadores et al. [2] have added support for reasoning for minimal RDFS rules in the distributed RDF storage system 4store. They include backward-chaining into the basic retrieval operation in a way very similar to the method presented in [29]. However, they avoid additional retrieval operations by synchronizing all schema information between the participating nodes using a dedicated node. While being able to deliver impressive scalability in experiments, the need for synchronization of schema information jeopardizes the adaptability and the robustness of their approach.

The same limitation applies to the work of Weaver et al. [34], who present a method for parallel RDFS reasoning. It is based on replicating all schema triples to all processing nodes and randomly partitioning the ABox, ignoring triples that extend the RDF Schema. The approach generates duplicates.

Hogan et al. [35] follow a pre-processing approach to scalable reasoning based on a semantics-preserving separation of terminological data. They create a set of stand-alone "template" rules formed from integrating the TBox into the reasoning rules. These rule sets are saturated with dependent rules and indexed for quick access, all aimed at a one-pass calculation of the full closure. They claim that their approach is distributable by distributing said rule sets to multiple nodes. While the template rules use a very similar notion as the reasoning operation presented here, their separation and template generation is based on the entire TBox being present at some point.

V. Conclusion and Future Work

We have explained how swarm self-organization is an interesting candidate to solve the challenges on the way towards web-scale storage, retrieval and reasoning on semantic data. Swarm algorithms already come with many of the properties desired for such systems: They are able to scale to an arbitrary number of operations, they exhibit robustness against failure and can adapt to almost any environment. In this paper, we have investigated our research question of whether swarm-based approaches are useful in creating a large-scale distributed storage and reasoning system. To this end, we have explained how the operations for the storage of new data, for the retrieval of data, and for the reasoning operation can be implemented according to the principles of swarm intelligence, in particular the foraging and brood sorting methods used by ants.

Since the non-deterministic mode of operation within these simulated swarms inhibits a formal proof of our approach, we have shown a stochastic analysis and experiments with black-box tests, where the behavior of the system is compared against a theoretical optimum. The experiments for the storage and retrieval operations measured the number of hops inside the storage network taken to retrieve any single triple. Results showed an almost perfect recall rate and—over several repeated experiments—an at least linear scaling behavior over the number of participating nodes. We have further compared these results with an optimal routing algorithm that always finds the perfect path inside the storage network. The comparison with our experimental results showed a two-fold increase in hops for our approach, which is acceptable in most cases. Thus, we assume our storage and retrieval operations will scale. To answer our research question, swarm-based approaches are indeed useful in creating a distributed storage and reasoning system for Semantic Web data, and we have shown the general feasibility and efficiency of our approach.

The goal of all reasoning operations on Semantic Web data is the generation of inferred statements. Thus, we compared the number of new statements generated by our swarm-based approach to reasoning against the number of inferred

statements calculated with a conventional stand-alone reasoner as a gold standard. These results showed the expected anytime behavior, where sound inferences are generated over time, approximating closure.

By comparing our approach to the related work in distributed storage and reasoning for Semantic Web data, we made two observations: First, distributed reasoning is a heavily-researched topic, answering the need we have identified in our introduction. Second, a wealth of methods ranging from MapReduce to random interactions is employed with impressive results, each either mainly focusing on completeness or on performance. However, these methods still have to be integrated with sufficient capabilities for storing new data and querying both the explicit and inferred statements, which has so far not been achieved in a fully decentralized way. We have presented a novel approach, where the reasoning operations re-use already present data structures for efficiency in a fully distributed way without any replication.

From our experiments that were aimed to show the potential of swarm algorithms for large-scale semantic storage, we are convinced that this idea is promising and merits further research in many directions.

V. Acknowledgments

We would like to thank our anonymous reviewers for their insightful and constructive comments. This research has been partially supported by the "DigiPolis" project funded by the German Federal Ministry of Education and Research (BMBF) under grant number 03WKP07B.

References

[1] C. Prehofer and C. Bettstetter. (2005, July). Self-organization in communication networks: Principles and design paradigms. *IEEE Commun. Mag.* [Online]. 43(7), pp. 78–85. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1470824

[2] M. Salvadores, G. Correndo, S. Harris, N. Gibbins, and N. Shadbolt, "The design and implementation of minimal RDFS backward reasoning in 4store," in *Proc. 8th Extended Semantic Web Conf. (ESWC)*, Heraklion, Greece, G. Antoniou, M. Grobelnik, E. Simperl, B. Parsia, D. Plexousakis, P. D. Leenheer, and J. Pan, Eds. Springer-Verlag, May 29–June 2, 2011.

[3] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Comput. Intell. Mag.*, vol. 1, no. 4, pp. 28–39, Nov. 2006.

[4] G. D. Caro and M. Dorigo. (1998). AntNet: Distributed stigmergetic control for communications networks. *J. Artif. Intell. Res.* [Online]. 9, pp. 317–365. Available: http://dx.doi.org/10.1613/jair.530

[5] W. Vogels. (2008). Eventually consistent. *ACM Queue* [Online]. 6(6), pp. 14–19. Available: http://doi.acm.org/10.1145/1466443.1466448

[6] L. Rutkowski, *Computational Intelligence—Methods And Techniques*. New York: Springer-Verlag, 2008.

[7] M. Mamei, R. Menezes, R. Tolksdorf, and F. Zambonelli. (2006, Aug.). Case studies for self-organization in computer science. *J. Syst. Arch.* [Online]. 52(8–9), pp. 443–460. Available: http://linkinghub.elsevier.com/retrieve/pii/S1383762106000166

[8] H. Mühleisen, A. Augustin, T. Walther, M. Harasic, K. Teymourian, and R. Tolksdorf. (2010). A self-organized semantic storage service, in *Proc. 12th Int. Conf. Information Integration and Web-based Applications and Services (iiWAS2010)*. ACM, pp. 357–364 [Online]. Available: http://portal.acm.org/citation.cfm?id=1967542

[9] K. Dentler, C. Guéret, and S. Schlobach, "Semantic web reasoning by swarm intelligence," in *Proc. 5th Int. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*.

[10] P. Hayes. (2004, Feb.). RDF semantics. World Wide Web Consortium, Recommendation REC-rdf-mt-20040210 [Online]. Available: http://www.w3.org/TR/rdf-mt/

[11] E. Prud'Hommeaux and A. Seaborne, "SPARQL query language for RDF," World Wide Web Consortium, Recommendation REC-rdf-sparql-query-20080115, Jan. 2008.

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*. Oxford: Oxford Univ. Press, 1999.

[13] H. Mühleisen, T. Walther, and R. Tolksdorf. (2011). Multi-level indexing in a distributed self-organized storage system, in *Proc. IEEE Congr. Evolutionary Computation (CEC)*, pp. 989–994 [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=5936494

[14] H. Mühleisen, T. Walther, and R. Tolksdorf. (2011). Data location optimization for a self-organized distributed storage system, in *Proc. 3rd World Congr. Nature and Biologically Inspired Computing (NaBIC)*, IEEE Press [Online]. Available: http: //hannes.muehleisen.org/NaBIC2011-muehleisen-s4-movement.pdf

[15] H. Mühleisen, "Query processing in a self-organized storage system," in *Proc. VLDB2011 PhD Workshop, co-located with 37th Int. Conf. Very Large Databases (VLDB)*, 2011.

[16] A. Schlicht and H. Stuckenschmidt. (2010). Peer-to-peer reasoning for interlinked ontologies. *Int. J. Semantic Computing (Special Issue on Web Scale Reasoning)*, pp. 1–31 [Online]. Available: http://ki.informatik.uni-mannheim.de/fileadmin/publication/Schlicht10p2p.pdf

[17] P. Obermeier, A. Augustin, and R. Tolksdorf, "Towards swarm-based federated web knowledgebases," in *Proc. Workshops der wissenschaftlichen Konferenz Kommunikation in verteilten Systemen* 2011, vol. 10.

[18] D. Fensel and F. van Harmelen. (2007). Unifying reasoning and search to web scale. *IEEE Internet Comput.* [Online]. 11(2), pp. 94–96. Available: http://doi.ieeecomputersociety.org/10.1109/MIC.2007.51

[19] D. Fensel, F. van Harmelen, B. Andersson, P. Brennan, H. Cunningham, E. D. Valle, F. Fischer, Z. Huang, A. Kiryakov, T. K.-I. Lee, L. Schooler, V. Tresp, S. Wesner, M. Witbrock, and N. Zhong, "Towards LarKC: A platform for web-scale reasoning," in *Proc. 2nd IEEE Int. Conf. Semantic Computing (ICSC)*. IEEE Press, 2008, pp. 524–529.

[20] D. Peleg and U. Pincas, "The average hop count measure for virtual path layouts," in *Proc. DISC: Int. Symp. Distributed Computing*, LNCS, 2001.

[21] S. Tang, H. Wang, and P. V. Mieghem. (2008). The effect of peer selection with hopcount or delay constraint on peer-to-peer networking, in *Networking* (Lecture Notes in Computer Science Series, vol. 4982), A. Das, H. K. Pung, F. B.-S. Lee, and L. W.-C. Wong, Eds. Springer-Verlag, pp. 358–365 [Online]. Available: http://dx.doi.org/10.1007/978-3-540-79549-0_31

[22] A. Fronczak, P. Fronczak, and J. A. Hołyst. (2004, Nov.). Average path length in random networks. *Phys. Rev. E* [Online]. 70(5), pp. 056110+. Available: http://dx.doi.org/10.1103/PhysRevE.70.056110

[23] A.-L. Barabasi and R. Albert, "Emergence of scaling in random networks," *Science*, vol. 286, no. 5439, pp. 509–512, Oct. 1999.

[24] A. Harth and P. Buitelaar. (2009). Exploring semantic web data sets with VisiNav, in *Proc. 6th Annu. European Semantic Web Conf. (ESWC2009)*, Poster Session [Online]. Available: http://sw.deri.org/2009/01/visinav/eswc-poster/visinav-poster.pdf

[25] G. Qi, J. Pan, and Q. Ji. (2007). Extending description logics with uncertainty reasoning in possibilistic logic, in *Proc. Symbolic and Quantitative Approaches to Reasoning with Uncertainty*, pp. 828–839 [Online]. Available: http://www.springerlink.com/index/j5102m4034j04235.pdf

[26] S. Kotoulas, E. Oren, and F. Van Harmelen. (2010). Mind the data skew: Distributed inferencing by speeddating in elastic regions, in *Proc. 19th Int. Conf. World Wide Web (WWW2010)*, ACM, pp. 531–540 [Online]. Available: http://portal.acm.org/citation.cfm?id=1772745

[27] D. Battré, F. Heine, A. Höing, and O. Kao. (2006). On triple dissemination, forward-chaining, and load balancing in DHT based RDF stores, in *Proc. 4th Int. Workshop Databases, Information Systems and Peer-to-Peer Computing (DBISP2P2006)* (Lecture Notes in Computer Science Series, vol. 4125), G. Moro, S. Bergamaschi, S. Joseph, J.-H. Morin, and A. M. Ouksel, Eds. Springer-Verlag, pp. 343–354 [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71661-7_33

[28] Q. Fang, Y. Zhao, G. Yang, and W. Zheng, "Scalable distributed ontology reasoning using DHT-based partitioning," in *Proc. 7th Int. Semantic Web Conf. (ISWC2008)*, 2008, pp. 91–105.

[29] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. (2010). RDFS reasoning and query answering on top of DHTs, in *Proc. 7th Int. Semantic Web Conf. (ISWC2008)*. Springer-Verlag, pp. 499–516 [Online]. Available: http://www.springerlink.com/index/V6440538394785H6.pdf

[30] E. Oren, S. Kotoulas, G. Anadiotis, R. Siebes, A. Ten Teije, and F. Van Harmelen. (2009). Marvin: Distributed reasoning over large-scale Semantic Web data. *J. Web Semantics* [Online]. 7(4), pp. 305–316. Available: http://linkinghub.elsevier.com/retrieve/pii/S1570826809000444

[31] J. Urbani, S. Kotoulas, E. Oren, and F. van Harmelen. (2009). Scalable distributed reasoning using MapReduce, in *Proc. 8th Int. Semantic Web Conf. (ISWC)*, Springer-Verlag, pp. 634–649 [Online]. Available: http://www.springerlink.com/index/M44432748XT110PJ.pdf

[32] J. Urbani, S. Kotoulas, J. Maassen, F. Van Harmelen, and H. Bal. (2010). OWL reasoning with WebPIE: Calculating the closure of 100 billion triples, in *Proc. 9th Int. Semantic Web Conf. (ISWC)*. Springer-Verlag, pp. 213–227 [Online]. Available: http://www.springerlink.com/index/2581664J64961667.pdf

[33] H. Horst. (2005). Combining RDF and part of OWL with rules: Semantics, decidability, complexity, in *Proc. 4th Int. Semantic Web Conf. (ISWC)*, pp. 668–684 [Online]. Available: http://www.springerlink.com/index/366474250nl35412.pdf

[34] J. Weaver and J. Hendler. (2009). Parallel materialization of the finite rdfs closure for hundreds of millions of triples, in *Proc. 8th Int. Semantic Web Conf. (ISWC)*. Springer-Verlag, pp. 682–697 [Online]. Available: http://www.springerlink.com/index/77X71125037K6583.pdf

[35] A. Hogan, J. Pan, and A. Polleres. (2010). SAOR: Template rule optimisations for distributed reasoning over 1 billion linked data triples, in *Proc. 9th Int. Semantic Web Conf. (ISWC)*, vol. 1380, pp. 337–353 [Online]. Available: http://www.springerlink.com/index/M6144754NM475404.pdf