Configuring a Self-Organized Semantic Storage Service

H. Mühleisen, T. Walther, A. Augustin, M. Harasic & R. Tolksdorf

Freie Universität Berlin
Department of Computer Science – Networked Information Systems Group
Königin-Luise-Str. 24/26, D-14195 Berlin, Germany
{muehleis,twalther,aaugusti,harasic}@inf.fu-berlin.de, tolk@ag-nbi.de
http://digipolis.ag-nbi.de

Abstract. Scalability requirements for semantic stores lead to distributed hardware-independent solutions to handle and analyze massive amounts of semantic data. We use a different approach by imitating the behaviour of swarm individuals to achieve this scalability. We have implemented our concept of a Self-organized Semantic Storage Service (S4) and present preliminary evaluation results in order to investigate to what extent the performance of a distributed and swarm-based storage system is dependent on its configuration.

1 Introduction and Motivation

Most Semantic Web applications require a semantic store, a specialized database for semantic data storage and analysis. Data for such applications is becoming more and more available, which leads to increased performance and scalability requirements for semantic stores. While considerable amounts of semantic data have been successfully processed on a single computer, distributed hardware-independent solutions are necessary to handle and analyze the massive amount of semantic data expected to become available in the near future.

Distributed storage poses two main questions: Where should an arbitrary data item be stored, and how should a specific stored item be located and retrieved efficiently. Many systems use a central catalog server, others maintain overlay network structures to answer these questions. We propose a different approach, where no node in the storage network has any distinct functionality, and where the tasks of data distribution and retrieval are performed by autonomous processes imitating the behaviour of swarm individuals, which have been shown to accomplish astonishing tasks using strictly local knowledge, limited memory, a limited set of rules, and a simple yet scalable way of passing information to other individuals.

This concept has been researched and simulated before as we will show in the following section, in this paper we focus on the presentation of central configuration parameters controlling the behaviour of our swarm-based system. We present preliminary evaluation results in order to answer our research question:

To what extent is the performance of a swarm-based system dependent on single configuration parameters?

To answer the research question in a setting as realistic as possible, we have implemented central parts from our concept of a Self-organized Semantic Storage Service (S4) and deployed this implementation on our lab network consisting of 150 virtual machines. For every relevant configuration parameter identified, a number of test runs was performed using different settings for this parameter. Test runs were comprised of storing a fixed dataset the system, and then testing whether the stored data could be retrieved efficiently from an arbitrary node.

The remainder of this paper is structured as follows: First, in Section 2, we introduce our relevant previous work and then continue with describing various approaches in distributed semantic storage and analysis. Section 3 then describes the basic concepts for retrieval and storage of data within our swarm-based S4 system with special focus on the relevant configuration parameters. Using our implementation, Section 4 presents our preliminary evaluation results both for storage and retrieval of a LUBM test data set. Finally, Section 5 concludes this paper by discussing evaluation results and describing our next steps.

2 Previous and Related Work

The need for distributed storage solutions emerges from the inherent limitations present on every stand-alone computer system. For those distributed solutions, two general approaches can be followed: Centralized network structures rely on systems orchestrating storage operations between storage nodes, while decentralized structures make no conceptual distinction between nodes, thus eliminating the single point of failure.

For the distributed decentralized storage of semantic information, various concepts such as Edutella [13], RDFPeers [2], GridVine [3] or YARS [9] have been proposed. They make use of Peer-to-Peer (P2P) technology to create an overlay network to store and retrieve semantic information in a distributed way.

Apart from mere storage, reasoning is also a crucial requirement for a semantic storage system. However, the support for reasoning is limited at best in the proposed concepts. So far, distributed reasoning has been attempted using different distribution techniques: Urbani et al. employ MapReduce to achieve reasoning over a very large amount of semantic data [18], Oren et al. use distributed hash tables (DHTs) in their MaRVIN reasoning system [15], and Dentler et al. rely on swarm intelligence for scalable reasoning [4].

Swarm intelligence has been identified to be a powerful family of methods by Bonabeau et al. [1]. Different applications of this family were described by Mamei et al. [10]. Menezes and Tolksdorf applied swarm intelligence to a distributed tuple space built to implement the Linda coordination model [6]. They introduced basic concepts of ant colony algorithms that are suitable for tuple storage and retrieval [11]. Tolksdorf and Augustin then applied this idea to distributed RDF storage and used a syntax-based similarity metric to cluster syntactically similar resources on neighboring storage nodes. Their concept was evaluated using simulation runs [16]. A similarity metric based on semantic similarity measures and aimed at achieving the clustering of related concepts was introduced in [17]. Harasic et al. proposed a different similarity metric using a hash function and also contributed an implementation architecture and evaluation results from a first prototype [8].

3 Self-Organized Semantic Storage Service - Concepts

In this section, we introduce basic concepts and operations of our Self-Organized Semantic Storage System (S4). The S4 system is a distributed semantic storage system. Triples are stored on a number of nodes that have been added to a storage network by configuration. This network does not contain a single central component, all a node has to know in order to join the network is the address of an arbitrary member. Each node maintains a list of neighbor nodes it is connected to. This list is built by a simple bootstrap algorithm. The algorithm simply asks all known nodes for other nodes, and tries to connect to them. The connection is successful, if the involved nodes have not yet reached their configurable neighbor limit. The process is repeated until a node has reached a minimum amount of neighbors, which is typically set to half the neighbor limit. It is expected that a higher neighbor limit leads to a better system performance, as a higher connectivity in the network reduces the average hop count to find a single node, and thus improves overall response time.

S4 offers two basic storage and retrieval operations, which are exported over an API available on every node. For both operations, custom adaptions of ant colony algorithms are employed. These algorithms have been described in [12], but we will outline the basic ideas and introduce the relevant configuration parameters as well as their expected effects below:

3.1 Retrieval

The system is able to locate information by a single key using a method of foraging found in the behaviour of several species of ants. Searching is performed by following virtual pheromone trails left behind by previous operations.

These trails are located to each connection to another node, thus by checking all present pheromones on a single node, an operation is able to determine which node to visit next, where it can access the triples stored there. This process is repeated until either a match was found or the configured maximum hop count has been reached. If a match was found, the path taken is tracked back in order to spread pheromones for the current key to be used by subsequent operations.

Pheromones are volatile due to the dynamics of the stored triples. For example, if a triple is deleted the pheromones leading to it should also disappear. Since manual removal would require broadcast messages through the entire network, their intensity decays by a configurable percentage per time unit. The following behaviour is expected for different decay rate settings: A smaller decay rate allows pheromones outlive the duration of few operations, and also leads to more triples being found.

For every retrieval operation, only a subset of the potential matches is to be returned. This is due to the intended use of our system by applications and also the expected size of the storage network. If the amount of data stored is exceeding a certain amount, it may be not feasible to return *all* matching results. Instead the user is asked to specify time an result set limit for the operation. Retrieval operations are terminated if one of these limits is reached.

3.2 Clustered Storage

A basic concept is the clustering of the stored triples by their similarity, which is intended to lead to triples with similar keys being stored on the same or neighboring storage nodes. This is achieved by a similarity metric and the "Brood Sorting" ant-based clustering algorithm. Storage operations are designed to store new triples on a node or in a neighborhood where similar triples are stored, according to the similarity metric. The new triples are taken on a path through the storage network similar to the retrieval operations and every node is checked for similar triples. If a sufficient amount of similar triples are found, the new triples are stored on the current node. Additional effort is invested to move misplaced triples to a node or a neighborhood where similar triples are stored, a special move operation performs this task by picking up triples not fitting on a node and moving them to another location. For information on the concrete similarity measures, we refer to the related work introduced in section 2.

3.3 Reasoning Support

The system performs forward-chaining assertional reasoning based on the knowledge that is held in the store and writes the inferred triples back into it. This section provides an overview of the basic concepts for reasoning used in the

semantic store. However, we do not not focus on this aspect yet, as the underlying storage layer has to reach a stable state first. A detailed discussion of the theoretical underpinnings has been given in [14]. The introduced basic storage operations cannot always guarantee correct results, reasoning on top of our system also trades away completeness for the degree of scalability we aim to achieve. Inferred statements are retrieved using the described read operations, there is no systemic differentiation between explicit and inferred triples.

In S4, terminological and assertional triples are stored together, i.e. TBox and ABox of the description logic are separated only conceptionally, not physically. For every terminological axiom that is inserted in the TBox a reasoning process is started. This process applies the axiom to matching assertions within the ABox, following the virtual pheromone trails through the network for the localization of the fitting triple clusters. Once a match is found, the resulting assertion is derived and then written back in the store in order to make it available for retrieval operations.

As an example we consider an example axiom from [14]:

$$ta := professor \sqcap researcher \sqsubseteq seniorresearcher$$

a part of the TBox. This means that every resource that is an instance of $professor$ and $researcher$ is also an instance of $seniorresearcher$.

In order to generate the inferred axioms, a retrieval operation for the triples matching the different predicates of the expression is executed. In this case the first step is to look for instances of the type *professor*. In a second step the corresponding axiom, that defines the matching instance to be also of the type *researcher* is looked for, again following the pheromone trails in the store. If the process identifies a matching instance, the resulting axiom to define the instance to be also of the type *seniorresearcher* is inferred and written back using the standard storage process as described.

Since all axioms for the reasoning process are retrieved from the store, the reasoning process is subject to the probabilistic influences of the swarm algorithms. Because of the constantly expanding data base in the store and its decentralized nature the reasoning itself is a continuous process. Thus, the inferral of new axioms can take some time as well as there is no guarantee for a certain inferred axiom to be retrievable at a certain time.

RDFS Inferencing To give an example of our approach on reasoning, we will present our implementations for a selection of RDF Schema (RDFS) inferencing rules in the following. For each rule, we present the task given to the reasoning operations, which then move through the storage network to fulfill their respective tasks. A similar approach has also been followed in [4], where swarm in-

dividuals are used to locate seldom-visited parts of an RDF graph. In our case, however, the storage network possesses the ability to find the path to nodes where triples matching the inferencing rules are stored, thus making a far larger amount of nodes possible.

For any property, a domain and a range can be defined. If a property has a concept as its domain, every resource annotated with this property is an instance of this concept. Range is very similar: If a property has a concept as its range, every resource referred to by this property is an instance of this concept. The formal definition for `rdfs:domain` and `rdfs:range` is given as follows:

```
(?x ?p ?y), (?p rdfs:domain ?c) -> (?x rdf:type ?c)
(?x ?p ?y), (?p rdfs:range ?c) -> (?y rdf:type ?c)
```

For our reasoning operation, domain definitions are evaluated using several steps. The following list shows the required steps to evaluate this rule within our swarm-based system.

1. Read a `rdfs:domain` statement on the local node in the form `(?p rdfs:domain ?c)`, bind `p` and `c` to values from the matching statement.
2. Using `p` as lookup key for routing, move to the next node
3. Locate all statements in the form `(?r1 p ?r2)`, bind `r1` to values from the matching statement.
4. Create new statements of the form `(?r1 rdf:type c)` for all matches.
5. Write new statements to the storage network.
6. Continue with step 2 until no more matches have been found for a configurable number of steps.

Range definitions are evaluated using the same method, but with `(?r2 rdf:type c)` as new statement in step 4.

3.4 Optimizations and System Behaviour

In order to efficiently compare the new triples with a potentially large amount of locally stored triples, the statements are again organized into local clusters [12]. The same local clustering algorithm which is based on the agglomerative hierarchical clustering method is used to limit the amount of pheromones present on each neighbor connection. Our clustering algorithm is configured by a limit for the maximum number of local clusters. A higher number of local clusters is expected to increase the accuracy of the global clustering, and hence the performance of the retrieval and storage processes.

Read operations are restarted, until a user-defined timeout or maximum result count is reached. If a read operation has been successful in locating triples matching its pattern on a particular node, results are sent back to the host the

query originated from. Since the notion of our similarity measures and global clustering supports the assumption of fitting triples being available on nearby nodes, the read operation is continued on the neighboring nodes.

The decisions on which node to go to next or whether new triples should be stored on the current node are influenced by various random factors in accordance to the basic principles of swarm-based algorithms. This leads to non-deterministic behaviour of the entire system. Thus, this approach can only be verified by simulations or test runs. In the conceptual phase, one can only make educated guesses about the influence single configuration parameters would have on the behaviour of the entire system.

3.5 Advantages of the Swarm-Based Approach

In contrast to other distributed storage systems, S4 does not require a central catalog server nor an overlay network structure that is costly to maintain in the event of network topology changes. Network organization is decentralized and robust to changes, as node failures only affect the data stored on that very node and perhaps the nodes the failed one was connected to. The remainder of the potentially huge storage network is completely unaffected. Every node has sufficient local information to take all decisions required from them by the straightforward swarm algorithms, hence eliminating error-prone synchronization. The main advantage over deterministic solutions is the ability of the swarm algorithms to adapt to an ever-changing environment very well, whether a single node may be overloaded with data or requests, or the mentioned node failure issue: Swarm algorithms have the potential to handle these issues without significant overhead, while still being able to efficiently respond to the various requests. For example, triples with the RDF:type property describing the type of an resource occurs in approx. 20% of the triples in our test data set. If data distribution is determined by an hash function, all those triples will be stored on and retrieved from the same node, which will then be soon overloaded, if a lot of queries contain this property (which is the case). In our system, a node will slowly start to reject triples if it detects its load approaching a certain limit. This will lead to these triples being stored on other nodes, all using only local knowledge and status and no observer whatsoever.

4 Performance Evaluation

In this section, preliminary evaluation results of our current S4 development version are presented. We have implemented the S4 system as a distributed Java application [12] and deployed it onto a cluster of 150 virtual Linux nodes running on a server equipped with eight 2.6 GHz processors and a total of 64 GB

memory. For each test run, a predefined data set generated by the LUBM data generator [7] containing 1.3 million triples was written to the storage system. After a short cool-down period, a single query was sent to all cluster nodes. The amount of results returned as well as the time required to return those results was measured. In order to determine the influence of the relevant configuration parameters, a set of configuration files covering various settings for each of the relevant parameters was created and a full test run was performed with all configuration files. In particular, the influence of the following parameters conceptually influencing the swarm algorithms was evaluated:

- `CLUSTER_LIMIT` - The maximum number of local clusters allowed for triple storage and pheromone management
- `MAX_STEPS` - The maximum amount of hops between nodes a single operation is allowed to perform
- `NEIGHBOR_LIMIT` - The amount of neighbor nodes to connect to
- `DECAY_RATE` - The decay rate of the virtual pheromone trails per time unit

Due to the probabilistic behaviour of the algorithms employed in the S4 system, no two test runs yield entirely equivalent results. The results presented below were taken from one single test cycle with identical software versions containing 30 test runs, each with a different configuration. However, these results are regarded to be exemplary, as other test cycles showed comparable results, and the test runs presented here have been selected to show the influence of the single parameters as clearly as possible. For this preliminary evaluation, the parameters were considered to be independent, in an attempt to reduce the search space.

4.1 Storage Performance

The LUBM-10 dataset we used to evaluate the system performance is written into 189 files by the data generator, each containing around 7000 triples. We used the HTTP API on an arbitrary node to store each file sequentially into the S4 system. Obviously, not all triples were stored on the node the requests were issued to. Fig. 1 shows the distribution of the amount of triples stored per storage node using a value of 25 for the `MAX_STEPS` parameter.

The time required to write a single file of the test data set into the S4 system was measured. Box plots describing the statistical trends of the write time for all files in the different configuration tested are given in Fig. 2:

As expected, a higher `CLUSTER_LIMIT` leads to a larger amount of comparably expensive cluster maintenance operations, therefore the write times for

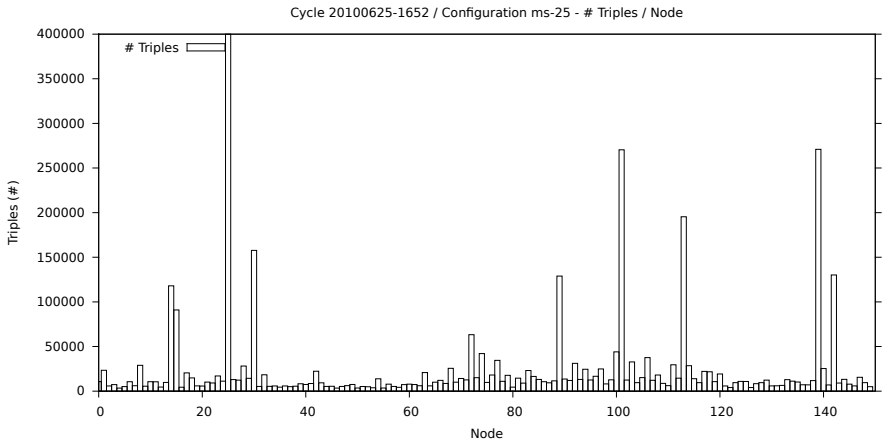

Fig. 1. Storage load distribution

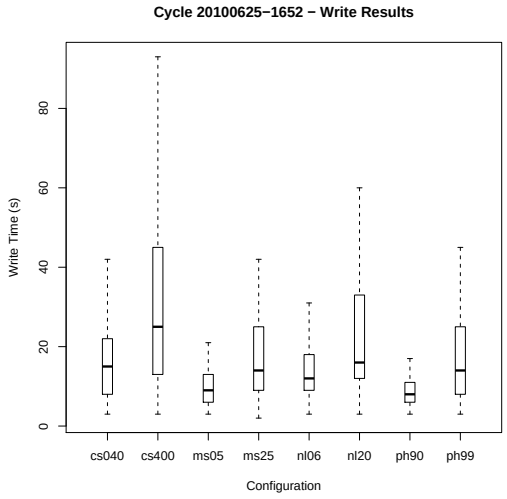

Fig. 2. Storage performance for LUBM-10 dataset files

configuration `cs400` with a cluster limit of 400 greatly exceed those of configuration `cs040` with a limit of 40.

The influence of the amount of steps to be taken, as configured by the `MAX_STEPS` parameter was tested in configuration sets `ms05` and `ms25` with 5 and 25 for maximum step number, respectively. As the transition of an operation from one storage node to another one is an expensive operation as well, a smaller value also leads to an improved write performance.

System performance was also expected to be influenced by the amount of neighbor nodes a single node connects itself to (`NEIGHBOR_LIMIT`). In contrast to our first expectation of a larger amount of neighbor nodes also distributing the load over more shoulders, configuration sets `nl06` and `nl20` with 6 and 20 as neighbor limits show a smaller neighborhood having a beneficial impact on the write performance. This may be due to the larger amount of possible paths that can be marked by the pheromones and thus missing pheromone data on some of the nodes.

A central configuration parameter for all systems based on ant foraging is the pheromone decay rate [5, p. 212 ff], thus the influence of this value over the parameter `DECAY_RATE` was also tested: Configuration `ph90` used a decay rate of 90, in which pheromone intensities are reduced by 10% every second, `ph99` employed only a 1% decay rate. Again, the results did not meet our expectations: Normally, longer pheromone endurance would result in better paths through the storage network, and thus reduced step counts, but the test result showed the opposite to be true for write operations. An possible explanation for this phenomenon might be the following: As pheromones on more frequently used paths are also updated more often, sub-optimal paths are then discarded sooner.

4.2 Retrieval Performance

To measure the performance for operations retrieving data from the S4 system, a SPARQL query was evaluated several times on each of the 150 storage nodes sequentially. The query contained a single triple pattern matching approx. 20% of the stored triples. Each node started the corresponding read operations within the S4 system, and collected the results from inside the storage network. The time limit for each query was set to five seconds, and the requested result set size was 1000 results. For each node, the amount of results returned as well as the time required to deliver the results was measured by the test driver. After three warm-up runs, five query executions were performed, and the average results determined the final result for a node in a configuration set. The result plots are structured as follows: The x axis contains all 150 storage nodes by their ID, the left y axis denotes the required time for each query ($+$ marker), and the right y axis describes the amount of triples returned ($\times$ marker). In general, a very low response time with a high amount of results returned are desirable for every storage node.

Fig. 3 shows the read results for different local cluster sizes, both for triple storage and pheromone clustering. The upper configuration used a `CLUSTER_LIMIT` value of 40, while the lower configuration used a value of 400. A higher amount of pheromone clusters leads to a more precise path selection and thus

node location, which reduces read times and increases result counts due to the smaller amount of hops required for a successful read operation.

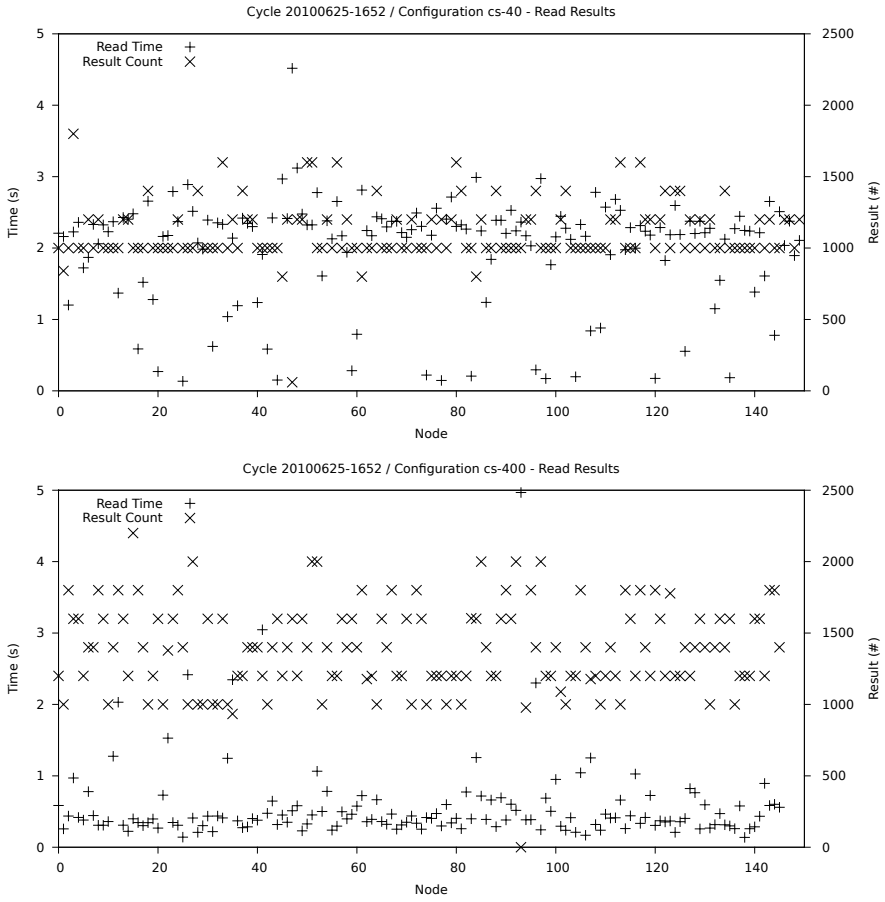

Fig. 3. Read results for `CLUSTER_LIMIT` parameter variations

The read operations are also greatly influenced by the amount of hops a single operation is allowed to take. Fig. 4 shows a test run with a `MAX_STEPS` setting of 5 on the top and 25 on the bottom. The upper configuration clearly shows a quick response time for the majority of nodes, but only the minimum amount of triples returned. This expected behaviour is confirmed by the bottom configuration: Not only are results delivered at a comparable speed, but the average amount of triples returned is also increased.

Contrary to in-memory executions of a swarm algorithm, the amount of steps allowed for a swarm individual has a huge impact on overall performance in a distributed system. This is due to the high costs of the transition operation between the storage nodes. Therefore, for any system built on swarm intelligence, this parameter has to be adjusted carefully.

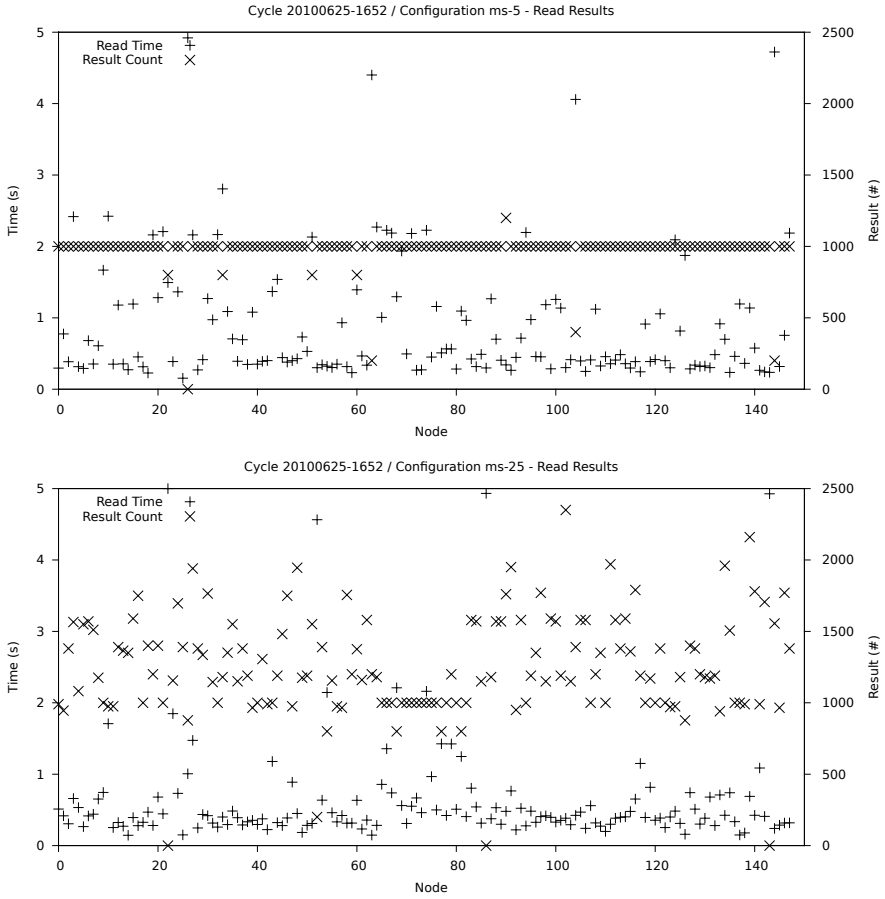

Fig. 4. Read results for `MAX_STEPS` parameter variations

Network setup from our bootstrapping algorithm is restricted to a limited amount of neighbor nodes. This behaviour is controlled by the `NEIGHBOR_LIMIT` parameter. In Fig. 5 two test runs with a neighbor limit of six on the top and a neighbor limit of 20 on the bottom are displayed. The configuration with the smaller limit performs better, the amount of triples returned is in gen-

eral higher, and the response times are consistently below one second with very few exceptions. This coincides with the observation of the read performance in the preceding section, where a smaller value for the neighbor limit also brought improvements in system performance. This may be due to a random error factor considered for the decision which neighbor to visit next. If there are more neighbor nodes, this error factor could have an increasingly disadvantageous influence, which will be evaluated in our further work.

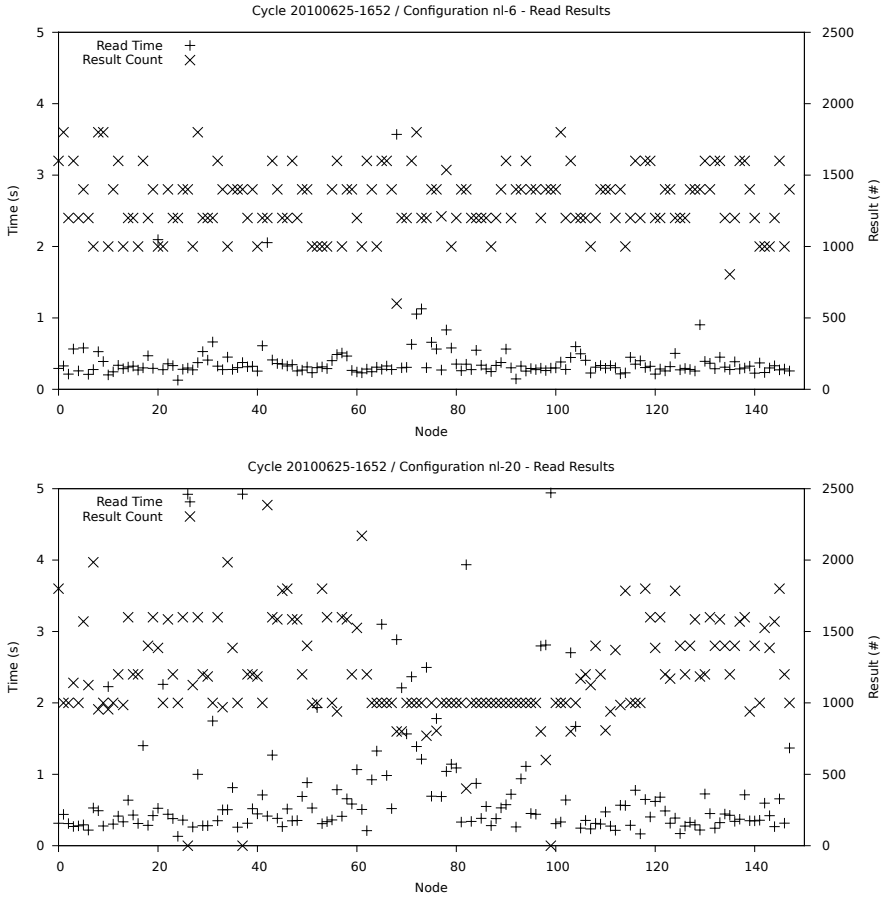

Fig. 5. Read results for `NEIGHBOR_LIMIT` parameter variations

The amount of decay in the intensity of the virtual pheromone also plays a role during read operations. Fig. 6 displays two test runs with a decay rate of 10% on the top and 1% on the bottom. As in the storage evaluation, the re-

sults again did not meet our expectations for the influence of this parameter. The test run with the increased decay shows very quick responses on nearly all nodes with a large amount of results, while the test run with the lower value shows considerably increased response times. This suggests that the introduction a mechanism to adjust this particular parameter is necessary for a distributed system based on swarm algorithms.

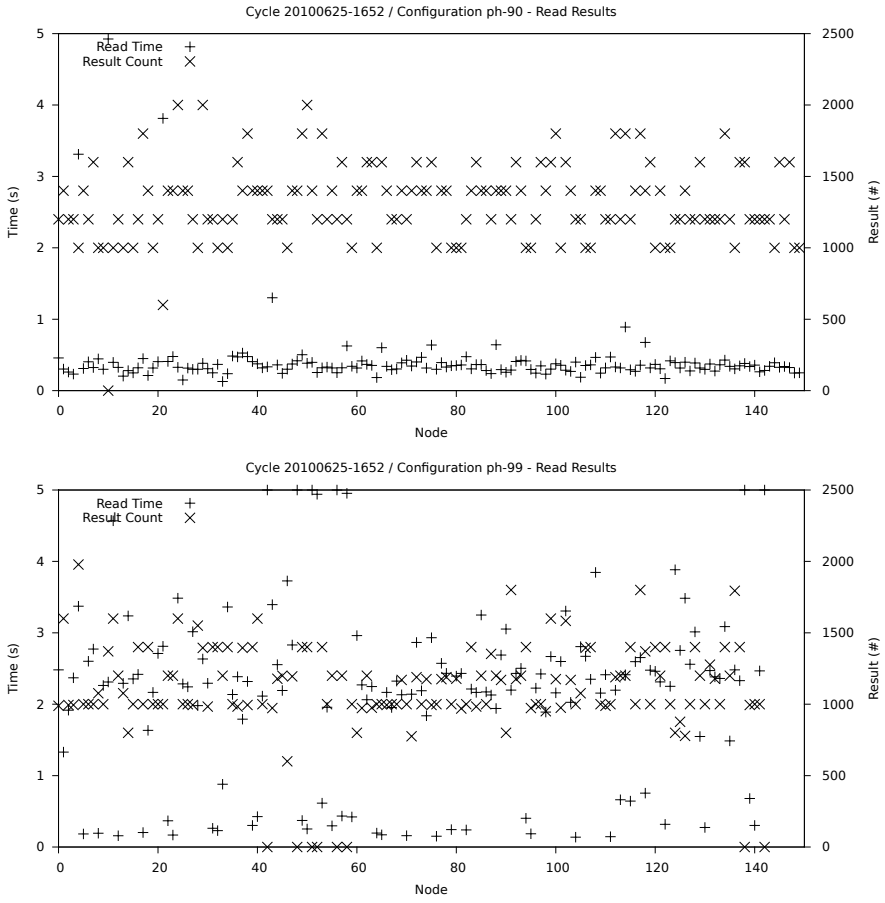

Fig. 6. Read results for `DECAY_RATE` parameter variations

5 Conclusion and Future Work

This paper started with outlining previous work in the application of swarm-based algorithms on distributed storage as well as distributed semantic storage

systems in general and current distributed reasoning approaches. We then continued introducing the relevant concepts of our Self-Organized Semantic Storage Service (S4), which uses self-organizing algorithms found in the behaviour of several ant species. A number of relevant configuration parameters for this system was identified and their expected impact on the system performance was described. Our preliminary evaluation of our implementation of the S4 system then showed the actual impact of the identified parameters in exemplary test runs using different values for those parameters in their configuration.

Our evaluation of basic storage and retrieval capabilities was set in an environment as close as possible to a real-world setting, as the desired performance of a swarm-based self-organizing system cannot be shown using formal proofs, but only by collecting statistical data. Therefore, simulations were not deemed to be satisfactory for our purpose. Successful test runs for a part of the configuration sets continue to show the general feasibility of building a larger-scale distributed system using swarm algorithms to facilitate self-organization. Tuning the various parameters was identified to be one of the main issues of swarm-based self-organizing systems, and tweaking the parameters to values where the storage network exhibits the desired results can be a tedious process, comparable to the work of a specialized database administrator. In some cases, heuristics could be used to adjust a particular parameter. However, special care has to be taken for these heuristics not to require global knowledge. This would contradict one of the basic concepts for swarm-based algorithms.

In our future work we would like to use our lab network to further advance our S4 implementation, and continue with evaluations using a variety of data sets, queries, network structures and configuration sets. We will implement the swarm-based distributed reasoning approach, and evaluate it as well. Further advancements of the S4 concepts are expected to be the design of additional auxiliary algorithms to support the basic algorithms, with the hope of achieving a system supporting its users by adjusting as many parameters as possible by itself.

Acknowledgments

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

References

1. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm Intelligence: From Natural to Artificial Systems. Santa Fe Institute Studies in the Sciences of Complexity Series, Oxford Press (July

1999)
2. Cai, M., Frank, M.: RDFPeers: a scalable distributed RDF repository based on a structured peer-to-peer network. In: WWW '04: Proceedings of the 13th international conference on World Wide Web. pp. 650–657. ACM, New York, NY, USA (2004)
3. Cudré-Mauroux, P., Agarwal, S., Aberer, K.: GridVine: An infrastructure for peer information management. IEEE Internet Computing 11(5), 36–44 (2007), http://doi.ieeecomputersociety.org/10.1109/MIC.2007.108
4. Dentler, K., Gueret, C., Schlobach, S.: Semantic web reasoning by swarm intelligence. In: Proceedings of Nature inspired Reasoning for the Semantic Web, ISWC 2009 (2009)
5. Dorigo, M., Stützle, T.: Ant Colony Optimization. The MIT Press, Cambridge, Massachusetts (2004)
6. Gelernter, D.: Generative communication in Linda. ACM Transactions on Programming Languages and Systems 7, 80–112 (1985)
7. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Sem 3(2-3), 158–182 (2005), http://dx.doi.org/10.1016/j.websem.2005.06.005
8. Harasic, M., Augustin, A., Tolksdorf, R., Obermeier, P.: Cluster mechanisms in a self-organizing distributed semantic store. In: Proceedings of Web Information System and Technologies, WEBIST 2010 (2010)
9. Harth, A., Umbrich, J., Hogan, A., Decker, S.: YARS2: A federated repository for querying graph structured data from the web. In: The Semantic Web, ISWC 2007, Busan, Korea, November 11-15, 2007. Lecture Notes in Computer Science, vol. 4825, pp. 211–224. Springer (2007), http://dx.doi.org/10.1007/978-3-540-76298-0_16
10. Mamei, M., Menezes, R., Tolksdorf, R., Zambonelli, F.: Case studies for self-organization in computer science. J. Syst. Archit. 52(8), 443–460 (2006)
11. Menezes, R., Tolksdorf, R.: A new approach to scalable linda-systems based on swarms. In: Proceedings of ACM SAC 2003. pp. 375–379 (2003)
12. Mühleisen, H., Harasic, M., Tolksdorf, R., Teymourian, K., Augustin, A.: A self-organized semantic storage service (2010), submitted to the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS2010), preprint available at http://digipolis.ag-nbi.de/preprint/iiwas2010-s4-preprint.pdf
13. Nejdl, W., Wolf, B., Qu, C., Decker, S., Sintek, M., Naeve, A., Nilsson, M., Palmr, M., Risch, T.: EDUTELLA: A P2P networking infrastructure based on RDF. Proceedings of the eleventh international World Wide Web Conference (Jan 01 2002), http://wwwconf.ecs.soton.ac.uk/archive/00000306/;http://wwwconf.ecs.soton.ac.uk/archive/00000306/01/index.htm
14. Obermeier, P., Augustin, A., Tolksdorf, R.: Towards swarm-based federated web knowledge-bases (2009), submitted to Nature inspired Reasoning for the Semantic Web (NatuReS09), preprint available at http://digipolis.ag-nbi.de/preprint/natures2009-sr-preprint.pdf
15. Oren, E., Kotoulas, S., Anadiotis, G.: MaRVIN: A platform for large-scale analysis of semantic web data. In: Proceeding of the WebSci'09: Society On-Line (March 2009)
16. Tolksdorf, R., Augustin, A.: Selforganisation in a storage for semantic information. Journal of Software 4 (2009)
17. Tolksdorf, R., Augustin, A., Koske, S.: Selforganization in distributed semantic repositories. Future Internet Symposium 2009 (FIS2009) (2009)
18. Urbani, J., Kotoulas, S., Oren, E., van Harmelen, F.: Scalable distributed reasoning using MapReduce. In: The Semantic Web - ISWC 2009, Chantilly, VA, USA, October 25-29, 2009. Lecture Notes in Computer Science, vol. 5823, pp. 634–649. Springer (2009), http://dx.doi.org/10.1007/978-3-642-04930-9