Data Location Optimization for a Self-Organized Distributed Storage System

Hannes Mühleisen, Tilman Walther and Robert Tolksdorf
Networked Information Systems Group
Freie Universität Berlin
Berlin, Germany
muehleis@inf.fu-berlin.de, tilman.walther@fu-berlin.de, tolk@ag-nbi.de

***Abstract*—Swarm-inspired algorithms allow the creation of complex systems that are scalable in many dimensions, adaptable to changing conditions, and robust against failure. These properties make them suitable for the challenges inherent in distributed storage systems. However, these swarm-based approaches reach their impressive performance by trading away correctness guarantees, occasionally leading to misplaced data items. In order to achieve consistent storage, there is a need for a constant optimization of the store's data structure. In this paper, we describe a fully distributed and scalable heuristic for the optimization of the location of stored data items within a distributed storage system based on the brood sorting method used by ants. We evaluate our heuristic using best- and worst-case test data sets to determine whether our location optimization method converges and whether it improves the location and organization of data inside a large-scale storage network.**

***Keywords*-Self-Organization, Swarm Intelligence, Ant Colony Optimization, Distributed Storage**

I. Introduction

The principles of self-organization and swarm intelligence can be applied to solve complex problems using very simple actors which can perform only a very limited set of actions with low complexity. Advantages are the independence from central components and the distribution of the evolving structures.

One problem of increasing complexity is the storage of data items in distributed environments. The main technical challenge in this area is scalability towards the amount of data elements that can be stored inside these systems and be retrieved from a single logical storage name space. While early storage systems used a single mainframe computer to save all data present in the system, federated systems were the first approach to distribute storage items on multiple physical computers while providing one logical data storage space. Here, one central computer holds a data structure that can resolve the location of a requested data item to a particular node. The next step was the introduction of fully distributed solutions, where central computers are not required anymore.

Adaptability to changing popularity of data items is an important challenge for fully distributed storage systems. For example, the Slashdot effect, where thousands of users access a web resource simultaneously, frequently disables even sophisticated storage architectures [1].

Therefore, a need for a fully distributed storage system able to scale to an arbitrary number of data items and able to adapt quickly to changing load patterns is present. We are in the process of developing an approach capable of fulfilling these requirements using a class of algorithms inspired by the behaviour of ants. The basic functionality of this *Self-Organized Semantic Storage Service* (S4) has already been described and validated through evaluation in our previous work [2]. However, as nature-inspired algorithms achieve scalability through operations which have a small probability of failure, we constantly work on improving our approach in order to achieve production readiness.

In this paper, we contribute a scalable heuristic and method for the optimization of the location of stored data items within our distributed store. This heuristic is based on the brood sorting method used by certain species of ants. We describe the rationale as well as the operations of this method in Section II. We evaluate our heuristic using best- and worst-case test data sets in Section III to determine whether our organization method does converge and improves the location and organization of data inside a large-scale storage network. Section IV gives an overview over related work in this area. Finally, Section V concludes our findings and gives an overview of future work.

II. Ant Colonies in Distributed Storage

Ant Colony Optimization (ACO) is the simulation of the behaviour of ants in order to solve problems efficiently. Solutions emerge through the loosely coordinated actions of large amounts of simulated individuals on a simulated landscape. Coordination between individuals is – consistent with nature – performed indirectly. Ants can deposit volatile markers (pheromones) on the landscape. A deposit will be made if an ant has encountered a solution. Markers can be detected by other ants which are then more likely to follow them. This represents a positive enforcement mechanism converging on global optimal solutions [3].

The main principles for ACO are simplicity, dynamism, and locality. Simplicity is what separates the swarm algorithms from multi-agent systems: swarm members have only a very limited capability for autonomous reasoning, their action rules have to be simple and straightforward. Dynamism is inherent in the landscape on which the ants move about,

pheromone paths change quickly, and items once found at a location might not be there forever. Locality is the principle that adds much-desired scalability to ACO: Every decision made by the ants has to be based on strictly local knowledge. No global mutable state of any kind has to be used in the ants' decisions.

First applications of ACO were performed to solve combinatorial problems such as the Travelling Salesman Problem (TSP), which stands as an example for a *NP*-hard problem solved efficiently using ACO [4]. For this class of problems, all swarm individuals and their landscape can be simulated on a single computer, and problem solutions are retrieved from the paths that emerge from the ants' movements.

A. Scalability in Storage Systems

The usability of a storage system is dependent on its scalability in many cases. Whenever a very large amount of data items is to be stored, or the amount of requests to the store exceeds the capabilities of stand-alone systems, a logical architectural choice is the distribution of the stored data over several physical computers.

If comparably few data items are served to a large number of requests, *replication* is the first method of choice. Here, data items are mirrored between several physical computers, and the requests for the stored data are distributed over the mirrors. However, for each update to the stored data set extensive replication operations become necessary. Also, the amount of data items that can be handled is dependent on the storage capacity of each single computer.

The second method is *federation*, where data items are distributed over several computers. Distributed storage poses challenges very similar to the routing problem. Data items are placed on a number of connected nodes, and each node is able to forward queries for the data items stored in the entire network to the node actually managing the data item. In federated systems, a distinguished node contains a data structure, which gives the storage location for each stored data item. Requests can be posed to the central node, and then access data items on the nodes they are stored on. However, as the amount of data items and requests increases, the central index node is subject to failures as well, impeding the operation of the entire system.

In a truly *distributed* system, no node alone is able to determine the location of all stored data items. Instead, this lookup is performed by a number of nodes, each contributing a small piece of the answer. Only this class of systems can reach a high degree of both scalability and robustness. The scalability of ACO in solving hard problems makes the approach a natural candidate for the determination of storage locations in distributed systems.

B. Operations of Swarm-based Distributed Storage

We proposed a distributed data store, which achieves scalability by using ACO-inspired algorithms to locate data items within a network of connected computers ("nodes"). Each node stores a subset of the data items present in the entire system. External processes can send requests to any node that is part of the storage network, where the request is converted to a virtual individual, which then can move through the storage network. These individuals take strictly local decisions, and do not rely on any global data structure. Efficient routing decisions are supported by virtual pheromones, which are deposited by successful operations. This way, virtual pheromone paths leading through the network to the different data items are formed.

This behaviour has been classified as "Deterministic backward ants and pheromone update" in [4]. This has a major advantage with the possibility to remove loops from the path taken before retracing it, usually using a variant of Floyd's algorithm for loop elimination from linked lists [5]. Loops are particular dangerous for ACO, since they can lead to undue amounts of deposited pheromones.

For application in distributed systems, ACO was lifted onto a new level: A set of computers (nodes) connected bilaterally using network technology is now considered to be the simulated landscape, and simulated ants are able to pass through the connections present between nodes. The simulation thus no longer is run on a single computer, where results can be retrieved easily by analyzing the landscape, but is rather part of a distributed system.

Since storage systems have to support the storage, the retrieval, and the removal of individual data items, pheromone values have to be differentiated between data items. For each data item stored, a separate ACO optimization takes place. We therefore introduced the notion of a *routing key*, which describes a defining property of the stored data items later to be used in retrieval. For example, if the storage system is to support file storage, a file name could be the routing key. Retrieval and deletion operations can be given the name of the file to be retrieved, and storage operations can use the name of the file to be stored to determine its storage location within the network. A simple mapping function maps an arbitrary routing key into a limited and fixed range $[0, 1]$. From there we can define our probabilistic routing policy for storage operations adapted from [6]. At each node k for each possible routing key d and all neighbor nodes n, P_{nd} gives the probability that node n is chosen as the next hop for the handling of the current storage operation.

$$\sum_{n \in \mathcal{N}_k} P_{nd} = 1,\ d \in [0,1],\ \mathcal{N}_k = \{neighbors(k)\}.$$

From the perspective of ACO, the routing key hence is the distinct value used to distinguish pheromone values. This poses a serious problem, as the amount of distinct data items is far larger than the amount of nodes in the network. Hence, we have developed a method to aggregate pheromone values into a limited amount of so-called "buckets", which are

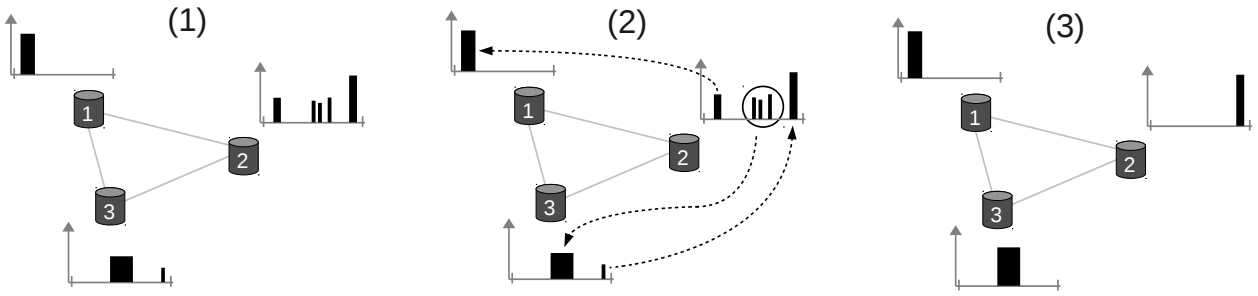

Figure 1. Bucket movement example

coarser entities of pheromone values [2]. Each bucket consists of a minimum and maximum mapping value, an average mapping value as well as an entry count. A maximum amount of possible buckets m is defined. For each new mapped routing key, a new bucket is created at first. Whenever the amount of possible buckets is exceeded, buckets are merged using a heuristic which merges closer and smaller buckets first [2]. This concept is also used in the organization of the data items stored on each node, because it improves the speed of local read operations and allows bulk transfers of data item groups as entire buckets.

The node on which a data item is stored is also a decision of the storage system. This decision can be based on a global law as for examples in hash-based distributed systems, where a hash function determines the node a data item is to be stored on. Main issues of global law data distribution are skewed key distributions and unbalanced data popularity. For example, if a hash function is used to determine storage locations, the quality of the distribution is directly dependent on the quality of the hash function and even distribution of routing keys. If the target range of the hash function is not used evenly, some storage nodes will have to store a larger quantity of data items, while some nodes will not store any data at all. Equally, if few data items are the subject of most retrieval operations, this will lead to a very unfair load distribution.

A more dynamic approach would be to determine the position of (new) data items using ACO, since this concept is not based on a global law in the aforementioned sense: Each swarm individual is independent in its decisions, in our case enabling the storage nodes to determine the next target for operations entirely by themselves. This can mend the problems of unfair data and load distribution, as the storage location of new data items can be set to an arbitrary storage node, with new pheromone paths forming on-demand.

C. Location Optimization by Re-Organization

An additional possibility is also the re-organization of already stored data items in order to improve the overall system performance. While a combinatorial solution would have no way in computing the optimal storage locations for all permutations of data items and storage locations, ACO may be able to achieve just that. However, a concept to determine the data items to be moved and the target nodes for these movement operations has to be created. While many forms of these concepts are conceivable, we propose a concrete approach using a simple heuristic inspired by the brood sorting method ants and bees use to keep their offspring organized [7]: Ants tasked with the organization of the brood will wander around in the nest randomly. When encountering larvae, they will pick up the most dissimilar one. They then carry it around the nest, and as they encounter other larvae, their inclination to put down the carried larva increases with its similarity to the larvae visible. This represents a distributed clustering algorithm that has already been successfully applied [8].

The need for storage location optimization arises from two sources: If new data items using previously unknown routing keys are stored they are simply placed on a random node. Also, due to the probabilistic operations of the systems, some storage operations do not reach the node storing data matching its routing key. These data items then have to be reorganized in order to be efficiently retrieved.

We have adapted the described brood sorting method to increase the coherence of stored data in a distributed storage system: In the method proposed in this paper, each host periodically generates a storage profile for the local persistent storage. This profile is then put onto a special type of ant, the "movement ant". The movement ant commences on a random walk through the network. On each visited node, the storage profile is compared to the local profile. The result of this comparison is a set of buckets, for which larger buckets also exist on the node where the move event originated from. This set is sent to the node where the movement ant originated and integrated into the data stored there.

This heuristic only increases the bucket size and decreases the amount of clusters present. Thus, if run for sufficient time, this process is expected to converge on an stable number of buckets. Through the merge of redundant buckets, ambiguous pheromone paths are made unnecessary, and the ability of the storage system to deliver all or most data items matching a particular query is also expected to increase. It has to be

noted that this heuristic does not require any global law, and thus is expected to be able to scale in regard to the storage network size.

Figure 1 gives an example for the application of the storage location heuristic in three stages. For three nodes, their storage profile is displayed as a graph giving the location of the buckets on the mapping range. For each box plotted, the left and right borders are plotted using the minimum and maximum bucket values. The height of each box is determined by the amount of data items stored in a bucket. We can now observe the evaluation of our heuristic using this (constructed) example. Panel 1 gives the original situation. Each node stores a set of buckets, but significant overlap exists between nodes. Panel 2 shows movement decisions: Because a larger and overlapping bucket exists on the remote node, a bucket is moved from node 2 to node 1, three buckets are moved from node 2 to node 3, and one bucket is moved from node 3 to node 2. The optimized situation is now displayed in Panel 3: Each node contains a smaller number of buckets, and no overlap in mapping values exists any more.

Reconsider Panel 1: There is an obvious bucket overlap in the leftmost bucket stored on nodes 1 and 2. If the ACO would have to optimize a path leading from node 3 to the data items with a routing key of 0.1 stored in these two buckets, the probability values would have to be shared between the two nodes, as in $P_{n_1 d_{0.1}} = 0.6$ and $P_{n_2 d_{0.1}} = 0.4$. In this case, 40 % of retrieval operations for the routing key 0.1 arriving at node 3 would be routed to node 2, and 60% to node 1. Either way, a significant share of data items stored for this routing key would not be found without additional requests or forwarding. However, for the optimized situation in Panel 3, only one pheromone path would be necessary for this routing key with $P_{n_1 d_{0.1}} = 1$. Now, all operations looking for data items matching this routing key from node 3 are forwarded to node 1, where all matching data items are stored. As shown by the example, storage optimization does enable more efficient pheromone path calculation and thus increases the performance of the entire storage network.

III. Evaluation

Our evaluation is set to show both the convergence as well as the scalability of our movement heuristic. We have extended our implementation of an ACO-based distributed triple store "S4" [2] to include the movement heuristic described in Section II-C. The extended implementation was deployed onto our testing cluster consisting of 100 virtual Linux servers. The test protocol was designed as follows: For storage network sizes ranging from 10 to 100 nodes, a data set containing 100,000 data items split up into 1,000 separate write operations was written into the storage network in four phases of 250 write operations each. After a phase was completed, four metrics were recorded in consecutive discrete samples:

1) Data items stored in the storage network
2) Movement operations that have *successfully* moved data
3) Amount of buckets in the storage network
4) Average number of data elements per bucket in the storage network

Sampling was continued until no successful movement operations were recorded for 10 samples, after which the next write phase was started. We have repeated our test protocol for two different data sets each containing 100,000 data items.

Two test data sets were generated for the purpose of evaluating the movement heuristic. Since the movement operation is relying on the bucket generation process in order to exchange storage profiles, the amount and distribution of possible buckets in the storage network is the main concern. Test data sets therefore represented two extremes for the movement heuristic. The first data set ("synthetic") contained only 200 distinct data identifiers that were used as input for the routing keys. The second data set ("random") included a distinct data identifier for each data item, hence 100,000 data identifiers.

Our movement heuristic is expected to converge as well as being able to optimize the synthetic data set with considerably less effort than the random data set. To show these two properties, two graphs have been produced for each test run: The graph on the left shows shows the amount of data items stored plotted against the amount of move operations per sample. The left vertical axis gives the scale for the amount of data items, the right vertical axis the amount of move operations. The horizontal axis gives the samples on a discrete scale. It should be noted that the time between each sample is not a constant factor, as the speed of data collection from the storage nodes depends on their processing load. The graph on the right plots the total amount of buckets inside the storage network against the average number of data items stored in each bucket, with the vertical axis being equal to the other plot.

Fig. 2 shows evaluation results for a storage network containing 100 nodes with the synthetic data set. The four write phases are represented by four large "steps" in the data item plot. The peaks in movement operations coincide closely with the increase of data items after each write phase. Also, movement peaks decrease in intensity over the four write phases. This is attributed to an increasingly improving storage situation due to optimizations. This can also be observed in the second part of the graph: Following each write phase, the growth of the total amount of buckets is reduced. Also, the average size of buckets is only decreased shortly and increased again as movement operations reorganize misplaced data items. For the synthetic data sets with its 200 distinct routing key the storage location optimization heuristic performs very well. Results for network sizes 10-90 are not given, since they produced very similar graphs.

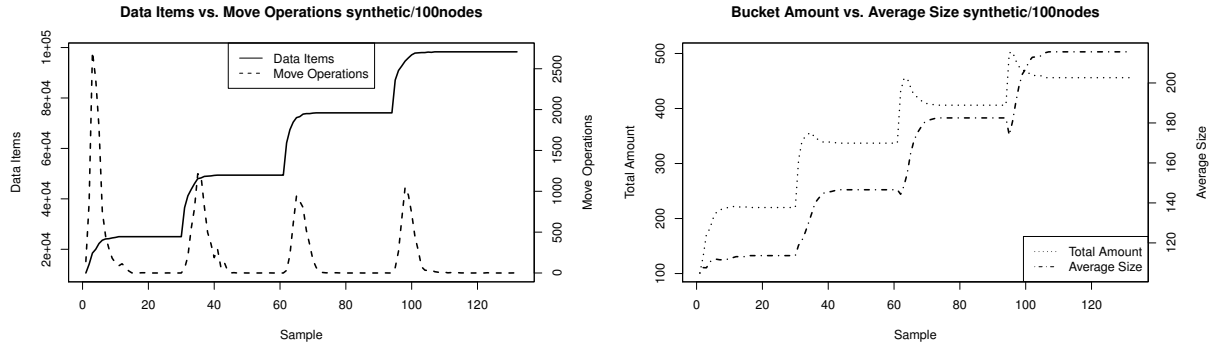

Figure 2. Evaluation results for 100 nodes and synthetic data set

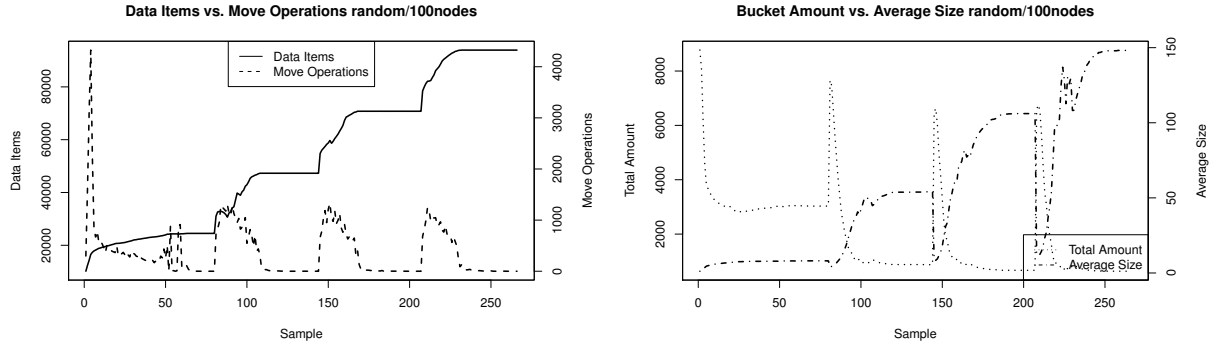

Figure 3. Evaluation results for 100 nodes and random data set

Result graphs for evaluation results using the random data set are plotted in Fig. 3. Again, results for different network sizes were too similar to be of interest. Two main differences to the results of the synthetic data sets can be spotted: First, the graph of the movement operations shows spikes of a different shape. This comes as no surprise, as a larger number of routing keys leads to a higher effort in location optimization. However, movement operations consistently converge, too. The second main difference is visible in the bucket statistics graph. The decrease in average bucket size and the increase in total bucket amount following write phases is much more intense. This also represents an expected behaviour and can be explained: For the synthetic data set, additional data items are likely to be able to use an existing pheromone trail due to the very limited number of routing keys. For the random data set, however, each data item has its own unique routing key, and pheromone paths are not as likely to be present. Hence, more data items are misplaced and have to be moved.

In conclusion, we have observed four main results of our evaluation of the self-organized storage location optimization heuristic:

1) Comparable results for all tested network sizes strongly suggest the scalability of our concept to the amount of storage nodes in a storage network.
2) We observed the convergence of the data distribution in the storage network to a stable state following write operations in every test run. This behaviour was expected and is caused by the heuristic only performing an action if bucket sizes are increased.
3) By plotting the amount of buckets against their average size for the entire storage network, we were able to observe a sustainable increase in bucket size. Apart from short decreases caused by the storage of additional data items, the average bucket size increased steadily over the four write phases. For both data sets the total amount of buckets did not increase linearly with the amount of items stored; for the random data set, additional data did even lead to less buckets than before due to bucket merging operations. The storage location optimization heuristic is therefore regarded to be able to improve the data distribution for arbitrary data sets.
4) Repetition of our evaluation for two different data sets with different amounts of routing keys enabled us to determine the impact of the distribution of routing keys in regard to the amount of elements stored. While the storage location optimization heuristic required considerably more move operations for the data set

with 100,000 routing keys, convergence and successful data reorganization was not hampered by this worst-case data set.

Our findings therefore confirm the overall scalability of our proposed re-organization heuristic based on brood sorting.

IV. Previous and Related Work

An example for the application of ACO to distributed systems was the *AntNet* concept, which uses ACO to solve the routing problem in packet-switched communication networks. Virtual ants are used to find routes to each popular network nodes. They move through the network following pheromone deposits from previous ant movements until they have arrived at their destination node. From there, they retrace their path back to their original node and deposit pheromones for their destination on the way. Here, ACO was lifted onto a new level: A set of computers (nodes) connected bilaterally using network technology was set to be the simulated landscape, and simulated ants are able to pass through the connections present between nodes. In AntNet, pheromone deposits are then used to create a probabilistic distance-vector routing table, to efficiently route data packets to their destination nodes- The system was able to match the performance of state-of-the art routing algorithms in extensive simulations. [6].

For distributed storage, Gelernter has proposed the *Linda Tuple Space* coordination model [9]. This model provides an abstraction of the possible operations necessary on a coordination medium. Here, tuples can be stored and retrieved from an opaque tuple space. However, the question how the Linda operations could be distributed onto multiple networked nodes in a scalable way sparked further research.

Jiang et al. proposed the *DTuples* concept, where a global-law-based Distributed Hash Table coordinates distributed operations [10]. However, the problems of unfair data distribution and load mentioned in Section II-B also appear here. A similar concept was used by Busi, Montresor and Zavattaro in *PeerSpaces*, where P2P technology is used to connect tuple spaces. However, they have discarded the data location transparency inherent in Linda by allowing the user to include the location information in user requests [11].

Menezes and Tolksdorf introduced the *SwarmLinda* concept, where ACO is used to locate data items in a distributed storage system. Virtual pheromone paths are laid by successful retrieval operations to assist future operations [12]. They continued in exploring the dimensions of adaptivity in such a system. The issue of the proper location of data was discussed on the common producer-consumer problem.

Viroli and Casadei have chosen another nature-inspired algorithm for their approach on providing a coordination model using simulated biochemical processes [13]. However, system performance is based on setting “chemical rules” and “reaction rates”, which requires a manual interaction.

Triple spaces for the storage of Semantic Web data encoded as Resource Description Framework (RDF) triples are a specialization of the very generic concept of a tuple space. Simperl, Krummenacher and Nixon proposed such a triple space as part of the *TripCom* project [14]. Here, the Linda principles were adapted to provide triple storage and retrieval from the space infrastructure. The data location transparency is again reduced with the possibility of specifying a concrete node from which tuples are to be retrieved.

Semantic Web scalability requirements led to a need for a large-scale distributed RDF triple store. Since storage of RDF triples can be considered a special case of the Linda triple space model, the adaption of SwarmLinda for RDF storage was a logical next step. The *RDFSwarms* concept by Tolksdorf and Augustin first used ACO algorithms for fully distributed RDF data storage and retrieval [15]. Building on top of RDFSwarms, the *Self-Organized Semantic Storage Service* (S4) extended the RDFSwarm concepts with triple and pheromone aggregation and provided a large-scale evaluation results confirming the feasibility of this concept [2].

V. Conclusion and Future Work

We have introduced Ant Colony Optimization (ACO) as the simulation of the behaviour of ants coordinated through the use of virtual volatile pheromones. Through simplicity, dynamism, and locality, ant colonies are able to converge on global optima while taking strictly local decisions, enabling scalability of the concept.

We described oour distributed storage system, which achieves scalability by using ACO-inspired algorithms to locate and place data items within a network of connected computers (“nodes”). We have further detailed our notion of a routing key, which describes a defining property of the stored data items later to be used in retrieval. We have used the routing key to describe distinct pheromone values. By pointing out the differences between distribution techniques based on global laws and swarm-based approaches, the location of data items emerged to be a variable and data reorganization as a possible solution to the issues of “unfair” data and load distribution. While many forms of data location algorithms are conceivable, we have contributed a concrete approach using a simple heuristic, where each host periodically generates a storage profile for the local persistent storage. This profile is then compared to the local profile on other nodes on a trip through the storage network, merging matching buckets together. The process was expected to converge on an stable number of buckets, improving the data distribution without requiring any global law.

Our evaluation was to show both the convergence as well as the scalability of our movement heuristic. We have observed four main results of our evaluation of the self-organized storage location optimization heuristic: The heuristic was able to perform well under network configurations ranging from 10 to 100 nodes, the amount of movement operations converged for every network size and for both data sets, and

the amount of buckets versus their average size matched our expectations. We therefore deem our storage location reorganization heuristic to be fit for usage in our and similar distributed storage systems and see no further issues impeding the scalability of our distributed storage system.

A. Future Work

The support for data items with more than one routing key is an area of further research. For example, should data items be tagged with a geographic location, queries for data items tagged with a particular position should be supported. This would provoke multiple levels of pheromone paths, and raise questions about trade-offs between indexing and reorganization efforts against their possible benefits. The means of processing queries including one or multiple of these indices are so far unknown for systems of this type. Also, our evaluation was limited to 100 storage nodes due to availability constraints. However, to further increase the confidence in our approach, we would like to run evaluations on far more nodes, possibly using cloud services.

Acknowledgments

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] A. Kaltenbrunner, V. Gómez, and V. López, "Description and prediction of slashdot activity," in *Fifth Latin American Web Congress (LA-Web 2007), 31 October - 2 November 2007, Santiago de Chile*, V. A. F. Almeida and R. A. Baeza-Yates, Eds. IEEE Computer Society, 2007, pp. 57–66. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/LA-WEB.2007.59

[2] H. Mühleisen, A. Augustin, T. Walther, M. Harasic, K. Teymourian, and R. Tolksdorf, "A self-organized semantic storage service," in *Proceedings of the 12th International Conference on Information Integration and Web-based Applications and Services*, ser. iiWAS '10. New York, NY, USA: ACM, 2010, pp. 357–364. [Online]. Available: http://doi.acm.org/10.1145/1967486.1967542

[3] T. Stützle and M. Dorigo, "A short convergence proof for a class of ant colony optimization algorithms," *IEEE-EC*, vol. 6, pp. 358–365, Aug. 2002.

[4] M. Dorigo and T. Stützle, *Ant Colony Optimization*. Cambridge, Massachusetts: The MIT Press, 2004.

[5] R. W. Floyd, "Nondeterministic algorithms," *Journal of the ACM*, vol. 14, no. 4, pp. 636–644, Oct. 1967.

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

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

[8] J. Handl and B. Meyer, "Ant-based and swarm-based clustering," *Swarm Intelligence*, vol. 1, no. 2, pp. 95–113, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11721-007-0008-7

[9] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, pp. 80–112, 1985.

[10] Y. Jiang, G. Xue, Z. Jia, and J. You, "DTuples: A distributed hash table based tuple space service for distributed coordination," in *GCC*. IEEE Computer Society, 2006, pp. 101–106. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/GCC.2006.41

[11] N. Busi, A. Montresor, and G. Zavattaro, "Data-driven coordination in peer-to-peer information systems," *Int. J. Cooperative Inf. Syst*, vol. 13, no. 1, pp. 63–89, 2004. [Online]. Available: http://dx.doi.org/10.1142/S0218843004000894

[12] R. Menezes and R. Tolksdorf, "A new approach to scalable linda-systems based on swarms," in *Proceedings of ACM SAC 2003*, 2003, pp. 375–379.

[13] M. Viroli and M. Casadei, "Biochemical tuple spaces for self-organising coordination," in *Coordination Models and Languages, 11th International Conference, COORDINATION 2009, Lisboa, Portugal, June 9-12, 2009. Proceedings*, ser. Lecture Notes in Computer Science, J. Field and V. T. Vasconcelos, Eds., vol. 5521. Springer, 2009, pp. 143–162. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-02053-7

[14] E. P. B. Simperl, R. Krummenacher, and L. J. B. Nixon, "A coordination model for triplespace computing," in *Coordination Models and Languages, 9th International Conference, COORDINATION 2007, Paphos, Cyprus, June 6-8, 2007, Proceedings*, ser. Lecture Notes in Computer Science, A. L. Murphy and J. Vitek, Eds., vol. 4467. Springer, 2007, pp. 1–18. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-72794-1_1

[15] R. Tolksdorf and A. Augustin, "Selforganisation in a storage for semantic information," *Journal of Software*, vol. 4, no. 8, pp. 798–807, 2009.