J. Parallel Distrib. Comput. 123 (2019) 90–99

Contents lists available at ScienceDirect

J. Parallel Distrib. Comput.

journal homepage: www.elsevier.com/locate/jpdc

Beyond scalability: Swarm intelligence affected by magnetic fields in distributed tuple spaces

Henrique D. Lima [a], Luiz A. de P. Lima Jr. [a,*], Alcides Calsavara [a], Henri F. Eberspächer [a], Ricardo C. Nabhen [a], Elias P. Duarte Jr. [b]

[a] Post-Graduate Program on Computer Science, Pontifícia Universidade Católica do Paraná, R. Imaculada Conceição, 1155, Curitiba-PR 80215-901, Brazil
[b] Department of Informatics, Universidade Federal do Paraná, P.O. Box 19018, Curitiba-PR 81531-980, Brazil

HIGHLIGHTS

- A scalable bioinspired method for data retrieval in distributed spaces is proposed.
- The behavior of data retrieving agents is affected by virtual magnetic fields.
- Simulation results in six different scenarios bring about the method strengths.
- Comparisons with previous approaches show how better performance was achieved.

ARTICLE INFO

Article history:
Received 15 February 2018
Received in revised form 16 June 2018
Accepted 3 September 2018
Available online 15 September 2018

Keywords:
Tuple space
Swarm intelligence
Virtual magnetic fields

ABSTRACT

Tuple Spaces have long been recognized as a simple and elegant model for parallel and distributed computing. This is mainly because of spatial and temporal uncoupling of system components, which simplifies inter-process communication as well as component inclusion and replacement. However, Tuple Spaces have shown scalability limitations when employed in large scale and highly demanding contexts. In order to deal with this problem, "bioinspired" techniques based on swarm intelligence including *SwarmLinda* and *Anti-Over-Clustering* have been proposed. This work shows that although these approaches do improve the system scalability, they end up producing an important degradation on tuple search performance, due to poor tuple placement. By applying the concept of "virtual magnetic fields" to swarms, a novel solution called *Magnetic SwarmLinda* is proposed. *Magnetic SwarmLinda* arranges tuples in expandable clusters of clusters (called "*magnetic clusters*") naturally providing load balancing among the supporting computing nodes. Simulation results show that the proposed strategy outperforms previous approaches in most scenarios.

1. Introduction

Distributed Tuple Spaces – or simply *Tuple Spaces* [1] – provide an associative shared memory computing model for process communication and coordination in distributed and parallel systems. Tuple Spaces have been employed in diverse areas such as the Internet of Things (IoT) [12], cloud computing [3,10], and to build large scale expert systems[1].

Data is stored in the distributed space in the form of *tuples*, which are ordered lists of fields of different data types, similar to the concept of *registers* in typed programming languages. Clients are either tuple consumers (that read or remove tuples from the space) and/or producers (that insert tuples into the space). The main difference between Tuple Spaces and other models is that tuples do not have explicit addresses. Searches in the tuple space are performed by association using *templates* which specify the "format" of the requested tuple. Templates are "incomplete" tuples in the sense that some of their fields may not hold instances of data, but only their type. For instance, the template ⟨*string*, 2018⟩ represents all 2-element tuples such that the first field is a string and the second is the integer 2018.

Processes communicate and synchronize with each other by inserting, reading and removing tuples from the Tuple Space. Because of this, the model provides *temporal uncoupling* (since providers and consumers do not need to be simultaneously "on line" in order to communicate) as well as *spatial uncoupling* (since the communicating processes can be arbitrarily distributed in the network).

The Tuple Space model was originally proposed and implemented in the context of the Linda programming language [6–8]. For this reason, it is generally identified with Linda itself.

* Corresponding author.
E-mail address: laplima@ppgia.pucpr.br (Luiz A. de P. Lima).
URL: http://www.ppgia.pucpr.br/~laplima (Luiz A. de P. Lima).

[1] www.gigaspaces.com.

https://doi.org/10.1016/j.jpdc.2018.09.004
0743-7315/

The language introduces two basic operations to insert and remove a tuple from the Tuple Space: `out` and `in`, respectively. The `out` operation takes a tuple as parameter, while the `in` operation takes a template as parameter. The `in` operation is synchronous and blocks the calling thread until a matching tuple is found in the Tuple Space.

Although tuple spaces allow the construction of distributed systems, the original systems (including JavaSpaces [20]), are based on a centralized component that is responsible for performing all the operations on the Tuple Space. Later fully distributed strategies were proposed, such as *SwarmLinda* [13] in which Tuple Spaces are implemented using swarm intelligence. In *SwarmLinda*, whenever an `out` operation is performed, a "tuple-ant" is instantiated with the goal of finding an adequate node to deposit the tuple. In order to improve the performance of future `in` operations, tuple-ants try to deposit their tuples in nodes with a greater concentration of similar tuples. In this way, *clusters* of similar tuples are created in the network and the paths leading to these nodes are naturally followed when searching for matching tuples. Although this approach does indeed improve the scalability of Tuple Spaces, it does not completely solve the problem because as the number of similar tuples increases, some nodes are likely to become overloaded. Moreover, overloaded nodes end up attracting more and more tuples, thus further aggravating the problem.

In order to solve that problem, [4] proposes an *Anti-Over-Clustering* strategy to overcome the excessive concentration of similar tuples in some nodes. In this approach, although a tuple-ant still tries to find a node with similar tuples in order to deposit its own, when the number of similar tuples gets close to a given threshold, the probability of depositing the tuple drops, thus decreasing the chance of creating over-sized clusters. Unfortunately, this strategy forces tuple-ants to carry on the exploration of the Tuple Space which turns out to consume more network and processing resources. Moreover, this strategy leads to the creation of clusters that are too scattered across the network, and this considerably worsens the performance of tuple retrieval.

In this work, we propose *Magnetic SwarmLinda*, a novel strategy based on swarm intelligence affected by "Virtual Magnetic Fields". The motivation is to both avoid overloaded nodes and excessive cluster dispersion thus improving tuple retrieval performance. Virtual Magnetic Fields are a distributed autonomic model originally employed to allow efficient application-level message routing – a message is delivered to the most suitable node – which is defined as the one with the greatest attraction force. The attraction force is a dynamic parameter defined by the application semantics. The model has been also applied to implement a fully distributed load-balancing mechanism [14] (in this case, the node's attraction force is inversely proportional to the node's load). In *Magnetic SwarmLinda*, however, magnetic fields act as magnetic shields that protect nodes from an excessive concentration of similar tuples. As a consequence, *clusters of clusters* (i.e., groups of neighboring nodes containing similar tuples) are created to store similar tuples.

In other words, the problem statement can be expressed as: develop a Tuple Space strategy that improves the performance of the tuple retrieval operations by avoiding excessive tuple concentration, and thus reducing agent competition, and therefore, node overloading. Furthermore, the proposed strategy also prevents excessive tuple dispersion by storing tuples in clusters of neighboring nodes, which significantly improves the performance of tuple retrieval. The proposed strategy *Magnetic SwarmLinda* was implemented and the results obtained show that its performance is superior to the other competing approaches except in scenarios where similar tuples are rare in the tuple space.

The remainder of the paper is organized as follows. Section 2 describes related work. The *Magnetic SwarmLinda* model is detailed in Section 3. In Section 4, experimental results are described including comparisons with the *Anti-Over-Clustering* approach. Conclusions follow in Section 5.

2. Related work – Distributed Tuple Spaces

In this section, some of the major *distributed* Tuple Space approaches are described. Arguably, WCL [16] is the first system based on geographically distributed Tuple Spaces. Although the basic components of WCL are centralized Tuple Spaces (each of which runs on a single node) WCL is capable of performing transparent migration of a Tuple Space from one node to another. This allows a specific Tuple Space to get closer to the nodes that issue most requests, thus reducing the communication latency. Migration can also occur when a given node becomes overloaded; in this case the strategy proves to be effective only when the extra load is due to the presence of several "small-sized" Tuple Spaces contributing to the load. When a single Tuple Space alone is the cause of the overload WCL does not work.

LInda in Mobile Environment (LIME) [15] is a Tuple Space implementation that uses mobile agents in a heterogeneous mobile network. Each agent provides a vision of the current set of available tuples. This set of available tuples expands or contracts as connections among hosts are established or dropped, respectively. The number of tuples in a Tuple Space can also change as agents move from one host to another. The distribution of tuples in LIME is quite particular. Each tuple inserted into the Tuple Space is stored in the host running the agent that was responsible for its insertion. When an agent migrates to another host, it carries along all its non-retrieved tuples. Because of this, when an agent cannot find a given tuple in its own portion of the shared Tuple Space, it keeps searching on other hosts of the Tuple Space until a matching tuple is found, this strategy results in unbound delays.

Dtuples [11] is a Tuple Space implementation based on a Distributed Hash Table (DHT). The authors argue that the DHT simplifies the communication among Tuple Space agents. Dtuples requires that the first element of a tuple be a string containing the tuple's "name". Names may not be unique, though. They are used to determine the location where tuples are stored. When retrieving a tuple, a name is again required to be the first field of the template so that the system may compute the location of a possible matching tuple. Unfortunately, this approach does not prevent unwanted concentration of tuples, since the hash function may produce a high number of collisions. Furthermore, the need for an additional element (the name) imposes undesired changes in the traditional Tuple Space interface.

Tupleware [1] is a Linda-inspired middleware that implements Tuple Spaces using a distributed algorithm to retrieve tuples. Tuples inserted by a process are stored in a local instance of the Tuple Space. This strategy may result in an excessive concentration of tuples on some nodes depending on the application communication pattern of specific applications. When a process requests a tuple which is not locally available, an application-transparent distributed search algorithm is initiated. This search is performed sequentially, starting with the nodes belonging to the Tuple Space that exhibit higher *success factors*. The success factor is computed from the historical rates of search hits on the nodes. In this way, the success factor is incremented for a particular node when the search is successful and decremented otherwise. This strategy may result in poor load distribution, since nodes with greater success factors will probably be highly requested.

SwarmLinda [13,18] is a Tuple Space implementation that uses swarm intelligence to provide scalability. Swarm intelligence is one of the several different approaches inspired by nature (also called "bioinspired" i.e. inspired by biological systems) that have been proposed to address problems in several areas of computer science. In SwarmLinda the tuple space is implemented through the interactions of very simple individuals inspired by ants. The system is fully distributed: the decisions made by these individuals

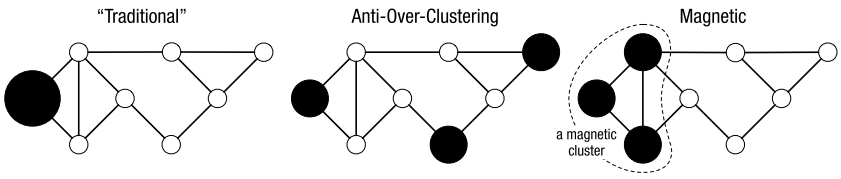

Fig. 1. Magnetic SwarmLinda distributes tuples in sets of neighboring nodes (called *magnetic clusters*), thus preventing ants from traveling very far when they are not found in a cluster node.

are based solely on local information, thus rendering the knowledge of the global system state unnecessary. This characteristic improves the performance of the system, as it prevents the intensive message exchanges that would be required to maintain the global state knowledge.

In SwarmLinda, whenever a *out* operation is performed, a mobile agent that implements an ant is instantiated to deposit a tuple in the tuple space. This agent, referred to as *tuple-ant*, is responsible for traversing the different nodes that make up the tuple space to find a suitable node to deposit the tuple. The criterion used by the tuple-ant to deposit a tuple is the number of tuples in the node that are similar to the tuple it is carrying. Thus, the probability to drop a tuple on a node is higher when there are many tuples similar to the tuple being carried. This behavior causes the formation of *clusters* of similar tuples in a given node.

As a *tuple-ant* moves from one node to the next, it deposits a "pheromone" that is specific for the type of tuple being carried on each visited node to mark the traversed path. Moreover, when a tuple is deposited on some node x, pheromone is also disseminated to x's neighbors.

Similarly, whenever an *in* operation is executed, a mobile agent is also instantiated. This agent, referred to as *template-ant*, is responsible for traversing the nodes of the Tuple Space to find a tuple that matches the template. When the *template-ant* finds a compatible tuple, it returns to its origin carrying that tuple. On its way back, a *template-ant* agent also marks that path by depositing pheromone on each visited node.

The strategy used by the *tuple-ant* and the *template-ant* to decide which nodes need to be visited is based on the concentration of pheromones that actually identify similar tuples. When an ant does not reach its goal in the current node, it obtains information about the number of similar tuples on that node and the intensity of the pheromone in neighboring nodes to determine the probability of moving on to each of these nodes. A *Time to Live* (*TTL*) variable is employed by the ants in order to prevent them from moving around indefinitely across the tuple space.

The creation of clusters of similar tuples prevents *template-ants* from traversing the whole tuple space as they search for a matching tuple. Instead, each *template-ant* concentrates its efforts in identifying the path (from the pheromone tracks) that lead to the cluster of interest. However, the higher the number of tuples of a cluster, the slower the retrieval process can be, due to the potentially large amount of ants being processed by the node. Moreover, overloaded nodes tend to become more attractive from the ant's point of view because of the high levels of pheromone leading to them.

In [5], the Anti-Over-Clustering strategy is proposed to avoid over-clustering of tuples in some nodes of SwarmLinda. This strategy is implemented by modifying the computation of the probability of a *tuple-ant* to deposit a tuple on a given node. The modification is based on a sigmoid function in which the probability to drop a tuple decreases when the visited node has a number of tuples higher than a threshold defined for the tuple space. The probability to drop a tuple on a node considering the Anti-Over-Clustering strategy is given by Eq. (1).

$$P'_{drop} = P_{drop} - \left(0.01 + \frac{P_{drop} - 0.01}{(1 + 0.5e^{-b(X-2m)})^2}\right) \quad (1)$$

In the expression above, variable X represents the number of tuples deposited on the node. Parameter m is computed based on X when the maximum value of the derivative curve is observed. The equation also takes into account the probability P_{drop} that a *tuple-ant* drop the tuple on a specific node (more details in Section 3.2). b depends on the maximum number of tuples allowed on a node. b gets smaller as the maximum number of tuples increases. Thus, the probability to drop a tuple is closer to the traditional approach when the node has a number of tuples that is well below the maximum. Similarly, the probability to deposit a tuple is minimum when the number of tuples is near to the maximum allowed. This change in the drop probability results in a delay of the time required for a *tuple-ant* to deposit its tuple in scenarios where there are over-clustered nodes. Therefore, the dropping is postponed so that the ant has the opportunity to find another node with similar tuples that is not overloaded. Unfortunately, this strategy forces ants to keep on exploring the Tuple Space; this movement by itself has a performance impact on the whole system. And since this approach is not concerned about the *location* where clusters are created, these clusters will be most probably all scattered over the network due to the random nature of their formation. This dispersion of clusters results in poor performance, particularly of the tuple retrieval process.

3. Magnetic SwarmLinda

Although the application of swarm intelligence in Tuple Spaces brings several improvements, it also gives rise to a critical problem: node overloading. In this section we present a novel strategy to deal exactly with this problem. Note that for ants in nature although pheromone trails constitute an important orientation mechanism, they are not the only information that they use. According to [19], other resources such as reference points, vibrations, gravity, solar compass and polarized light that are also used for orientation. Moreover, according to [2], magnetic fields may cause a shift in the orientation of certain ant species. In the present work we introduce a fully distributed strategy for implementing Distributed Tuple Spaces called *Magnetic SwarmLinda* which applies the concept of magnetic fields to affect the "normal" behavior of swarms with the goal of improving the performance of tuple retrieval operations, in particular, by preventing excessive concentration of tuples in a single node and, at the same time, avoiding the excessive dispersion of similar tuples across the network.

Fig. 1 illustrates the difference of the three approaches: traditional *SwarmLinda*, Anti-Over-Clustering, and *Magnetic SwarmLinda*. Black nodes contain tuples that are similar to the one being looked for and the "size" of the node represents the number of tuples of that type that the node holds. *SwarmLinda* was proposed to improve the scalability of earlier approaches, but unfortunately it leads to certain nodes to become overloaded with similar tuples. A growing number of tuples at a node has a direct impact on the scalability, as the search performance of that node will decrease and there is a limit of the number of tuples a single node can keep. Anti-Over-Clustering solved that problem, as it effectively avoids overloaded nodes, but unfortunately it created another problem by scattering similar tuples across the network and thus making

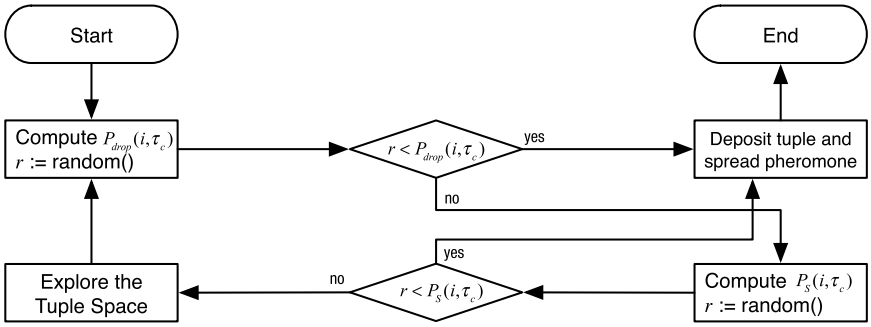

Fig. 2. The simplified behavior of *tuple-ants*.

Table 1
Description of the notations used in this paper.

Notation	Description
τ_c	Tuple carried by a *tuple-ant* or desired by a *template-ant*
$P_{drop}(i, \tau_c)$	Probability of dropping a tuple τ_c on the current node i
$C(i, \tau_c)$	*Concentration* at node i of tuples that are similar to τ_c
$P(\tau_c)_{i,j}$	Probability of moving from node i to node j considering τ_c
$Ph(k, \tau_c)$	Current amount of pheromone of node k considering τ_c
$M_L(i, \tau_c)$	*Magnetic level* of node i considering a tuple τ_c
$M_c(i)$	*Magnetic restriction* of node i based on its capacity
$F_M(i, \tau_c)$	*Magnetic force* of some node i considering a tuple τ_c
$P_S(i, \tau_c)$	Probability that a "strange behavior" would affect a *tuple-ant*

tuple retrieval a costly procedure, as large portions of the network have to be traversed to retrieve tuples of certain types. *Magnetic SwarmLinda* solves the problem by using magnetic clusters that improve the scalability not only by avoiding overloaded nodes but also by avoiding scattering similar tuples too far away from each other.

The *Magnetic SwarmLinda* model can be formalized as follows. Let $G = (V, E)$ be the network graph, where V is the set of nodes and $E \subseteq V \times V$ is the set of edges representing bidirectional links, each connecting a pair of nodes. Let $N = |V|$ be the number of nodes and $M = |E|$, the number of links of G. $NH(i)$ represents the set of direct neighbors of node $i \in V$ (i.e., $NH(i) = \{j : (i, j) \in E \vee (j, i) \in E\}$. The notation that will be used throughout this paper are described in Table 1.

The next sections detail the *Magnetic SwarmLinda* model. First, the behavior of *tuple-ants* (created by the `out` operation) is specified. Then, the equations that support the decision making process concerning tuple drop, path selection and magnetic interference are described in detail.

3.1. The `out` Operation and the Behavior of Tuple-Ants

When a Tuple Space receives a request to perform $out(\tau_c)$ (in which τ_c is the tuple carried by a *tuple-ant* to be inserted in the Tuple Space), the following steps are taken:

1. A new ant agent (a *tuple-ant*) associated with τ_c is created. This agent is configured with the maximum number of hops it can traverse set to the *TTL* (Time To Live) field. This *tuple-ant* is responsible for depositing τ_c on some node of the Tuple Space.
2. Let i denote the current location (i.e., the node it currently is at) of the *tuple-ant*. The *tuple-ant* then checks if many similar tuples have already been deposited on i. This information is used to decide whether τ_c should or not be deposited in the current node. The probability of dropping τ_c increases as the number of similar tuples grows (as described in Section 3.2).
3. If the *tuple-ant* decides to deposit τ_c on node i, then the only thing it has to do before dying is to mark its location by spreading its pheromone on i and on all surrounding adjacent neighbors (i.e, $NH(i)$). This process reinforces i as an adequate location for future similar tuples, so that other *tuple-ants* carrying similar tuples will have a greater probability choosing i.
4. If the *tuple-ant* decides *not* to deposit the tuple on i, then it must choose an adjacent node to go to. This choice is made stochastically based on information concerning the neighboring nodes (as detailed in Section 3.3).
5. As the ant moves to a new node, its gets older, i.e., the corresponding *TTL* is decremented. If $TLL = 0$, then the *tuple-ant* deposits the tuple on the current node spreading its pheromone as described in step 3 and then, it dies. On the other hand, if $TTL > 0$, then the *tuple-ant* keeps going as described in step 2 but now in a new node.

The behavior of *tuple-ants* (see flow-chart of Fig. 2) can be simplified by not checking the *TTL* field at all if the drop probability guarantees that the tuple will be deposited when $TTL = 0$. That is exactly what is done in the proposed model (as it will be shown in 3.2).

The probability of dropping tuple τ_c in the current node i – $P_{drop}(i, \tau_c)$ – is based on the tuple concentration on i (see Section 3.2 for details). After computing $P_{drop}(i, \tau_c)$, a *tuple-ant* generates a random number $r : 0 \leq r < 1$. If $r < P_{drop}(i, \tau_c)$, then the τ_c is deposited in the current node i and the corresponding pheromone is spread. Otherwise, the *tuple-ant* continues exploring the Tuple Space (as described in Section 3.3), unless it develops what we call a "strange" behavior that results from being affected by a strong magnetic field. The probability of exhibiting this behavior – $P_S(i, \tau_c)$ – is described in Section 3.5).

3.2. The drop probability

As mentioned in Section 2, the probability of dropping a tuple on the current node depends on the concentration of *similar* tuples on that node. A *similarity function* – $sim(\tau_A, \tau_B) \in [0, 1]$ – must therefore be defined so that the degree of similarity between two tuples τ_A and τ_B can be assessed. Although more sophisticated application-dependent functions could be employed, for the experiments of Section 4, the simple binary similarity function in Eq. (2) produced good results (see Section 4). In this case, tuples are considered similar if their numbers of fields and the respective types of each field match.

$$sim(\tau_A, \tau_B) = \begin{cases} 1, & \text{if } template(\tau_A) = template(\tau_B) \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

The *concentration* at node i of tuples that are similar to τ_c (the tuple that the agent is currently carrying) – $C(i, \tau_c)$ – is given by Eq. (3). $C(i, \tau_c)$ is computed by comparing τ_c with each tuple τ_s stored in $i \in V$.

$$C(i, \tau_c) = \sum_{\forall \tau_s \in i} sim(\tau_c, \tau_s) \quad (3)$$

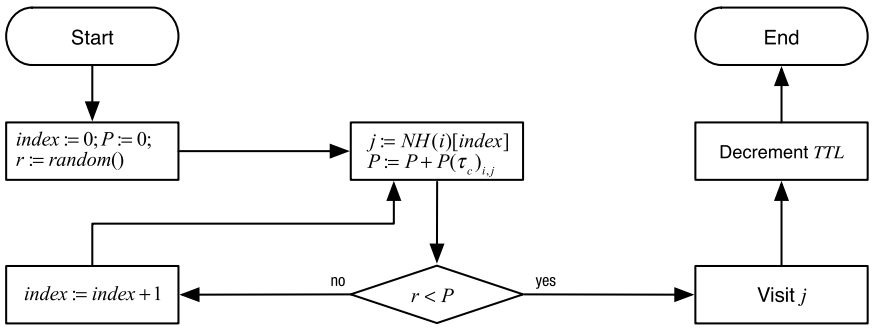

Fig. 3. How a *tuple-ant* explores the Tuple Space.

Therefore, $P_{drop}(i, \tau_c)$ which corresponds to the probability of depositing tuple τ_c on node i is given by Eq. (4). $\mu > 0$ is a predefined (small) constant used to guarantee that there will always be a (small) probability of depositing τ_c in i (i.e., μ prevents $P_{drop}(i, \tau_c)$ from being zero), even when there are no other similar tuples stored on i. In the simulation experiments described in Section 4, $\mu = 0.0001$.

$$P_{drop}(i, \tau_c) = \left(\frac{C(i, \tau_c) + \mu}{C(i, \tau_c) + \mu + TTL}\right)^2 \quad (4)$$

Notice as well that $P_{drop}(i, \tau_c)$ depends on the *TTL* value. The probability of dropping the tuple increases as the *TTL* decreases. Remember that *TTL* represents an upper bound of the number of hops that a *tuple-ant* traverses in order to drop a tuple. The *TTL* value prevents *tuple-ants* from moving indefinitely around the nodes of a tuple space thus overloading the whole system. In other words, the probability of dropping a tuple is 100% when the $TTL = 0$. Moreover, as mentioned above, the probability of depositing a tuple on a given node is directly proportional to the concentration of similar tuples on that node. As the influence of *TTL* on $P_{drop}(i, \tau_c)$ decreases as $C(i, \tau_c)$ increases this causes the formation of clusters of nodes that are close to each other all of them storing similar tuples.

3.3. Path selection

If a *tuple-ant* does not achieve its goal (i.e., dropping the tuple) on the current node, it needs to choose a neighboring node to visit so that it can carry on. In order to increase the chance of an ant achieving its goal, it should move along paths which many other individuals carrying similar tuples have previously traversed. Moreover, a *tuple-ant* must consider the number of similar tuples in a potential destination node. Eq. (5) defines the probability that an ant at node i carrying tuple τ_c moves to another node j, where $Ph(k, \tau_c)$ represents the current amount of pheromone corresponding to tuple τ_c in node $k \in V$.

$$P(\tau_c)_{i,j} = \frac{C(j, \tau_c) + Ph(j, \tau_c)}{\sum_{\forall n \in NH(i)}(C(n, \tau_c) + Ph(n, \tau_c))} \quad (5)$$

Fig. 3 shows how a *tuple-ant* explores the network. For each neighbor in set $NH(i)$, the *tuple-ant* compares a random value $r \in [0, 1)$ with P, which is the cumulative sum of $P(\tau_c)_{i,j}$ for the neighbor under evaluation ($j = NH(i)[index]$) and the previously evaluated neighbors. The next node corresponds to the neighbor for which P is greater than or equal to r. Notice that when all neighbors have been evaluated, P will reach 1.0. Therefore, the *tuple-ant* always chooses one neighbor to go to. When it finds some neighbor j that satisfies $r < P$, then it moves to node j. As mentioned above, as it moves it "gets older" by decrementing *TTL*.

3.4. Pheromone evaporation

The pheromone evaporation mechanism is essential to make the system adaptable, since trails leading to node regions that no longer have a significant concentration of tuples must disappear as time goes by. An "evaporation" mechanism prevents *tuple-ants* from having a chaotic behavior thus compromising the proper functioning of the whole system. Furthermore, the evaporation mechanism makes shorter paths more attractive, thus optimizing the amount of hops necessary for *tuple-ants* to reach the desired graph regions. Eq. (6) defines how pheromones of some node i fades out as time t passes by.

$$Ph_t(i, \tau_c) = Ph_{(t-1)}(i, \tau_c)(1 - \rho) \quad (6)$$

All system nodes decrease the amount of local pheromone according to the evaporation rate ρ ($\rho \in [0, 1]$). Nevertheless, ρ must not be so high as to prevent the exploration of new trails. On the other hand, if ρ is too small, the displacement of *tuple-ants* can be negatively affected by trails that lead to regions that no longer have similar tuples. It is important to highlight that all decision-making is based only on the current concentration of pheromone. Thus, it is not required to maintain information about previous concentrations levels. Thereby, for the sake of simplicity, parameter t will be omitted from $Ph_t(i, \tau_c)$ from this point on.

The model present up to this point suffers from scalability problems, since similar tuples tend to be deposited on the same node that rapidly becomes overloaded, causing a severe degradation on future tuple retrieval operations. In order to solve this problem, we introduce the concept of magnetic fields.

3.5. Magnetic Interference

Magnetic Interference is a mechanism to prevent the excessive concentration of tuples in a few (possibly overloaded) nodes. The "traditional" approach states that the probability that a new tuple is deposited on a given node is proportional to the number of similar tuples the node holds. This property improves the performance of tuple retrieval operations by guiding the search process to only a certain region of the tuple space. However, when there is an excessive amount of tuples stored in a single node, it is likely that the node will become a bottleneck. The overload is related to the fact that such a node holding a large number of tuples will most probably receive too many ants trying to retrieve and/or deposit tuples. This is particularly relevant for *template-ants* that produce a greater load when compared with *tuple-ants*, since the *template-ants* may have to perform several match-trial operations in order to find a tuple that meets the restrictions specified in the template.

The *magnetic level* that some node i is exposed to with respect to a given tuple τ_c is defined by Eq. (7).

$$M_L(i, \tau_c) = Max\{C(n, \tau_c) : n \in NH(i)\} \quad (7)$$

In other words, for a given node i, the magnetic level corresponds to the maximum concentration of tuples that are similar to τ_c in i's neighbors. M_L is used to determine the *magnetic force* (or *strength*) of some node i, designated by $F_M(i, \tau_c)$ (see Eq. (8)). This magnetic force is responsible for producing the so called "strange behavior" on a *tuple-ant*, causing it to drop the tuple in the current node earlier than normally expected.

$$F_M(i, \tau_c) = \frac{M_c(i) \times M_L(i, \tau_c)}{M_c(i) + M_L(i, \tau_c)} \tag{8}$$

The magnetic force affecting *tuple-ants* at node i tends to zero when there is no significant magnetic field exerted by i's neighbors. $F_M(i, \tau_c)$ depends on the *magnetic restriction* positive constant ($M_c(i) > 0$) which represents the maximum number of tuples that node i can store without being considered overloaded.

The probability of the "strange behavior" affecting a *tuple-ant* currently at node i carrying tuple τ_c is defined by Eq. (9).

$$P_S(i, \tau_c) = \frac{F_M(i, \tau_c) + C(i, \tau_c)}{\sum_{\forall n \in NH(i)}(C(n, \tau_c) + Ph(n, \tau_c)) + F_M(i, \tau_c) + C(i, \tau_c)} \tag{9}$$

A given node i that has an overloaded neighbor is under a high level of magnetic interference (computed using $F_M(i, \tau_c)$) and, consequently, *tuple-ants* at i carrying tuples similar to τ_c are very likely to exhibit the strange behavior. This prevents most *tuple-ants* (since the decision is stochastic) from reaching already overloaded neighbors. Because of this behavior, groups of nodes storing similar tuples (called *magnetic clusters*) tend to be formed. At the same time magnetic interference also prevents nodes from being overloaded.

4. Experimental validation

In order to evaluate the performance of the *Magnetic SwarmLinda* approach, an event-based simulator for multi-agent systems was implemented. The simulator is similar to *SimPy* [17], which is a process-based discrete-event simulation framework built with Python. In this section, results are presented for six different scenarios. In each scenario the performance of *Magnetic SwarmLinda* was compared both with "traditional" *SwarmLinda* and *SwarmLinda* with Anti-Over-Clustering.

The metric used to evaluate the performance is the average tuple retrieval delay, that is, the response time for executing an `in` operation. Time is measured in *Ideal Time Units* (**itu**s). Each **itu** represents the delay for a message to be delivered from one node to a direct neighbor. The choice of evaluating just the `in` operation is justified by the fact that, from the application point of view, the tuple insertion operation is instantaneous, since the `out` operation is asynchronous (i.e., it does not require the client to wait for its completion).

For simulation purposes, the magnetic constraint $M_c(i)$ was set to the "desirable" amount of tuples to be deposited on each node. This value was obtained by considering a uniform distribution of all tuples throughout the tuple space (the total amount of tuples deposited in tuple space is known in the simulation). So, $M_c(i) = T/N, \forall i \in V$, where T is the total number of tuples in the system (regardless their "types"), V is the set of nodes and $N = |V|$ is the number of nodes (as explained in Section 3).

In order to evaluate the performance of the proposed approach, a set of client processes was defined to perform operations on the tuple space. In the architecture, for each node $i \in V$, there is a client process that is responsible for inserting and retrieving tuples. Moreover, we assume that the network topology represented by graph G is connected.

Each client periodically executes tuple insertion and tuple retrieval operations alternately with a given probability. In order to measure the performance of tuple retrieval operations, clients only try to retrieve tuples that are already available in the tuple space, since application-dependent delays would be introduced if clients had to wait indefinitely till an unavailable tuple becomes available. It is important to highlight that these simulation parameters and architecture do not restrict neither the number nor the location of clients, nor the topology of the application (except that it must be connected). In other words, scenarios in which some nodes have multiple connected clients and others with no clients are both possible.

The number of possible *types* of tuples to be generated – designated by τ^t – as well as the *variety* of tuples of a particular type – designated by τ^v – are simulation parameters. τ^v represents the set of possible values for a given tuple field type. For instance, $\tau_A = \langle 10, \text{"}Bob\text{"}, 1.2\rangle$ and $\tau_B = \langle 5, \text{"}Alice\text{"}, 2.3\rangle$ are examples of two instances of the same type of tuple (that consists of an integer field, a string-typed field and a floating point field).

4.1. Simulation parameters and scenarios

For each scenario, random graphs were generated using NetworkX [9], a Python library for the "creation, manipulation, and study of the structure, dynamics, and functions of complex networks". The tuple space topology used is based on Watts–Strogatz's *Small-World Graphs* [21], which are known to represent the structure and dynamics of social, biological, and infrastructure networks. A random Small-World graph $G = (V, E)$ is generated by initially creating a ring with $N = |V|$ nodes, each one connected to its k nearest neighbors. Then, each edge $(u, v) \in E$ is replaced with probability σ by a new edge (u, w) where $w \in V$ is randomly chosen. The following values were used for all simulation runs: $\sigma = 30\%$ (so that the resulting topology will present a certain degree of randomness) and $N = 16$. The default value of k is 4.

For each simulation run, a different seed for the uniformly distributed pseudo-random number generator was used. Each client has the same probability of producing any of the τ^t types of tuples and their τ^v possible values when executing an `out` operation. Similarly, each client also has an equal probability of producing any type of templates when executing an `in` operation.

Initially, before retrieving any tuple with the `in` operation, clients execute `out` operations until T tuples have been deposited into the tuple space. This is needed so that the performance can be measured without interference from fluctuations that occur in the early stages of the formation of clusters and pheromone trails. Furthermore, the total amount of tuples in the tuple space T tends to remain unchanged, since client processes alternate executing `in` and `out` operations. Notice that this is done with the purpose of removing application-dependent delays (cases in which suitable tuples are not available in the system) from the simulation results that would introduce distortions of the results as mentioned above.

In order to be able to observe variations of the response time due to some nodes being overload, it is necessary to impose limits on the resources available at each node. For this purpose, parameter *OP* is defined to represent the number of operations that a node can process within an **itu**. These operations include: checking whether the tuple-template matches, selecting a neighbor to move the ant and deciding whether to drop or not a tuple. An excessive number of (similar) tuples stored in a node has a negative impact on the performance of the operations executed by *template-ants* to search for a tuple that matches a requested template.

The default values of the main simulation parameters are shown in Table 2.

In order to evaluate the performance of *Magnetic SwarmLinda* in different contexts, six distinct scenarios were used (see Table 3). Parameters were chosen in such a way to allow the evaluation of representative application profiles, e.g., varying the number of

Table 2
Default simulation parameters.

Parameter	Value	Description
k	4	the average number of edges per node
S	50	the number of simulation runs
ρ	20%	the evaporation rate (see Section 3.4)
τ^t	8	the number of types of tuples
T	100,000	the number of tuples deposited in tuple space
τ^v	1,000	the number of possible values for each type of tuple
D	1,000	the simulation duration in terms of number of `in` operations
OP	1,000	the maximum number of operations per **itu**
I	10	the time interval (in **itu**s) between consecutive operations

Table 3
Evaluation scenarios — In each scenario, parameters other then the ones in the second column are shown in Table 2.

Scenario	Varying Parameter	Parameter Values	Performance evaluation for
1	I	3, 4, 5, 6 and 7	high rates of operation executions
2	I	5, 15, 25 and 35	high and low rates of operation executions
3	τ^v	500, 1000, 1500 and 2000	different tuples of the each type
4	k	4, 6, 8 and 10	different average number of edges per node
5	T	10^5, 2×10^5, 3×10^5 and 4×10^5	different number of permanent tuples in the tuple space
6	τ^t	3, 4, 5, 6, 7 and 8	different number of types of tuples

types of tuples, the number of tuples of each type, the frequency in which operations are executed, the density of the tuple space topology, among other features. In each scenario, the proposed approach is compared with both the "Traditional" *SwarmLinda* and with *SwarmLinda* with Anti-Over-Clustering. Each scenario evaluates the impact of a single parameter. The other parameters remain at their default values shown in Table 2.

In Scenarios 1 and 2, the performance of the three approaches is evaluated for different load levels. In order to do so, the simulation parameter I that represents the time intervals between two consecutive operations is set to different values as shown in Table 3. The smaller the value of I, the greater the frequency of operations and therefore the heavier the load of the tuple space. Scenario 1 differs from Scenario 2 only with respect to the time intervals; in Scenario 1, the demand on the tuple space is heavier.

Scenario 3 was designed to evaluate the system response with retrieval operations trying to find rare tuples. The greater the value of τ^v, the larger the variety of possible values for each tuple field and the smaller the number of identical tuples (since the total number of tuples is constant).

In Scenario 4, all approaches are evaluated with respect to the density of the Tuple Space topology. With smaller k values, the network topology becomes less dense.

Scenario 5 was designed to investigate how the number of tuples T deposited on the tuple space during the warm-up phase affects the performance of the different approaches. Since clients alternate executing tuple insertion and removal operations, the total amount of tuples on the tuple space is nearly constant during each simulation run. This allows us to assess the performance of each approach for different numbers of tuples in the Tuple Space.

Finally, in Scenario 6, the performance of the tuple space is evaluated for different numbers of types of tuples. This allows us to measure the impact of different amounts of node clusters, since each cluster (in the *Magnetic SwarmLinda* model) is associated with a given type of tuple.

In addition to the evaluation scenarios shown in Table 3, another experimentation was executed using the parameters of Scenario 2 in order to evaluate how far a *template-ant* travels in each

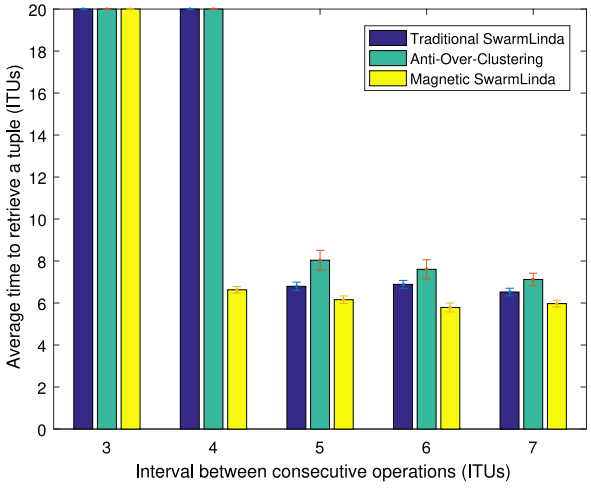

Fig. 4. Scenario 1 – Performance evaluation in a high demanding scenario.

of the three approaches. Thus, only the time spent for traversing the network during the execution of the tuple retrieval request is computed, leaving aside the time spent for processing the tuples. This metric is called the round-trip delay of a *template-ant*.

4.2. Simulation results

The results obtained for Scenario 1 are shown in Fig. 4. Scenario 1 aims at evaluating the average delays to retrieve a tuple from the tuple space when clients issue operations at very short time intervals (i.e., in a high demanding scenario). The confidence level for the results is of 95%.

From the results obtained, it is possible to identify the *saturation point* of all the approaches.

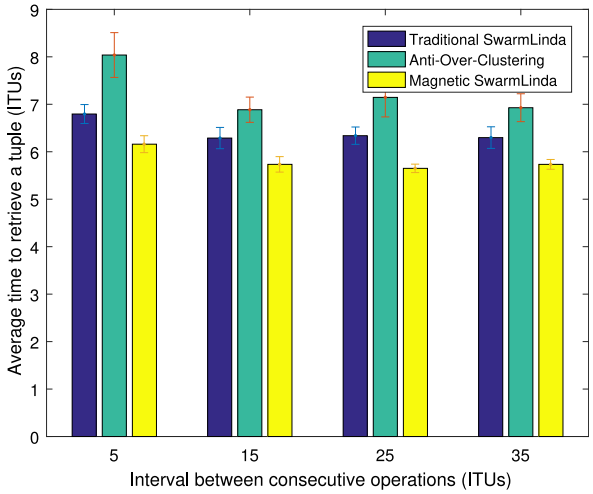

Fig. 5. Scenario 2 – Performance evaluation for different (non-saturated) loads.

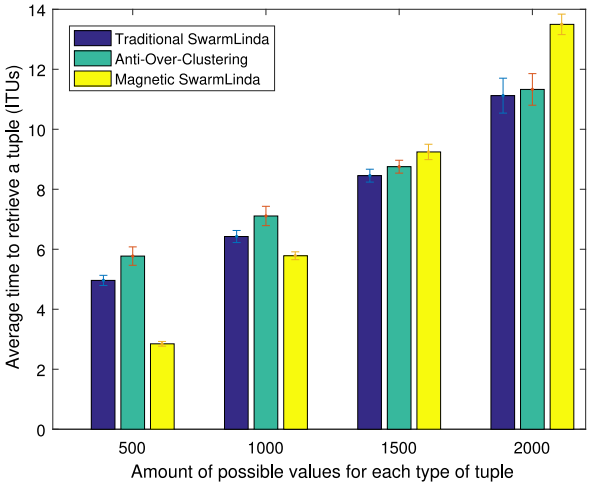

Fig. 6. Scenario 3 – Performance evaluation for different possible values for each type of tuple.

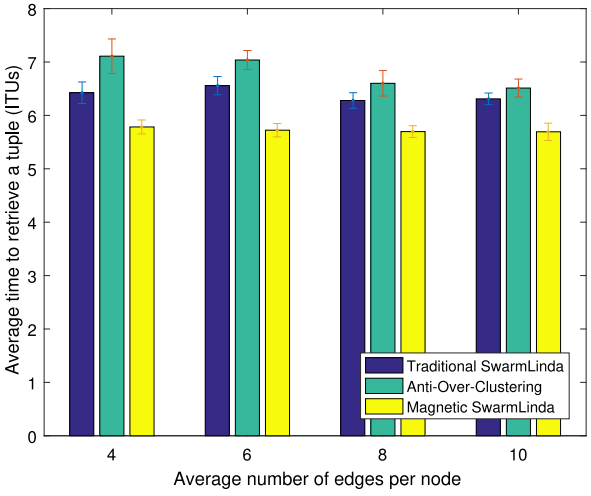

Fig. 7. Scenario 4 – Performance evaluation for different network densities.

This point is reached when the system starts taking longer and longer to execute tuple request operations. Below the saturation point the time interval between consecutive tuple request operations is shorter than the system's capacity of effectively executing the operations. Above the saturation point, the Tuple Space sustains the load, causing, therefore, a steady performance degradation at each new operation. The saturation point for the *Magnetic SwarmLinda* approach happens when $I = 3$. For the other two approaches (*SwarmLinda* and *SwarmLinda* with *Anti-Over-Clustering*), the saturation point occurs when $I = 4$.

In Fig. 4, the y axis has been "trimmed" at 20 in order to better show the behavior of the three approaches when $I \geq 4$. Furthermore, even for the traditional approach, when $I = 4$, the average tuple retrieval time tends to infinity as the simulation proceeds. This happens because with $I = 4$ the clients are inserting tuples every 8 **itu**s, while the mean time to recover a tuple is at about 20 **itu**s.

Performance results for Scenario 2 are shown in Fig. 5. Notice that the average delay to retrieve a tuple using the Magnetic approach was always smaller than the delay of the other two approaches when the tuple space is not saturated.

As expected, all approaches show faster response times when clients execute operations less frequently. However, the performance improvement is not so relevant since, unlike Scenario 1, the Tuple Space is not exposed to a load that could cause the saturation of the system.

Results for Scenario 3 (Fig. 6) demonstrate that the proposed approach performs well in scenarios where there is a relatively small number of possible values for a given type of tuple. This behavior is due to the fact that a small set of options will end up generating more identical tuples in the Tuple Space. Thus, the number of nodes that a *template-ant* needs to visit also tends to be small since there is a greater probability of finding compatible tuples in the group of neighboring nodes containing similar tuples formed as a result of the Magnetic approach. This group of nodes is a *magnetic cluster*.

On the other hand, Traditional SwarmLinda and the approach that employs Anti-Over-Clustering both present a similar performance in scenarios where there is a large number of possible values for a given type of tuple (notice that their confidence intervals overlap). This happens due to the small amount of identical tuples present in the Tuple Space, which forces *template-ants* to visit more nodes that belong to a magnetic cluster. As the number of similar tuples decreases, i.e., the number of rare tuples increases, the performance of the proposed strategy becomes worse that of traditional SwarmLinda and with Anti-Over-Clustering. This happens because, in order to find a rare tuple, more cluster nodes have to be visited.

The results obtained for Scenario 4 (Fig. 7) show that all three approaches can be considered stable in terms of the response delay to retrieve a tuple for different Tuple Space topologies. In the case of *Magnetic SwarmLinda*, this happens because even for a less dense topology in which nodes have a small number of neighbors, the likelihood of a *template-ant* to reach any magnetic cluster with a small number of hops is high because the magnetic cluster is composed of multiple nodes. Therefore, there is a high probability that there is a short path from any node in the Tuple Space to at least one node of some magnetic cluster.

Anti-Over-Clustering is the approach that benefits the most from a denser graph since a *tuple-ant* located in an overloaded node is more likely to find a path that leads to another node with many similar tuples and that is possibly not overloaded. Moreover, a *template-ant* looking for a compatible tuple will take advantage of the fact that there is a higher probability of finding nearby nodes with similar tuples. In this scenario, the traditional approach presents an intermediate performance.

In Scenario 5, Fig. 8 shows that *Magnetic SwarmLinda* is clearly better for different total amounts of tuples deposited in the Tuple Space. Furthermore, the average time to retrieve a tuple decreases very steadily using this approach. SwarmLinda and SwarmLinda with Anti-Over-Clustering presented a slower improvement as the

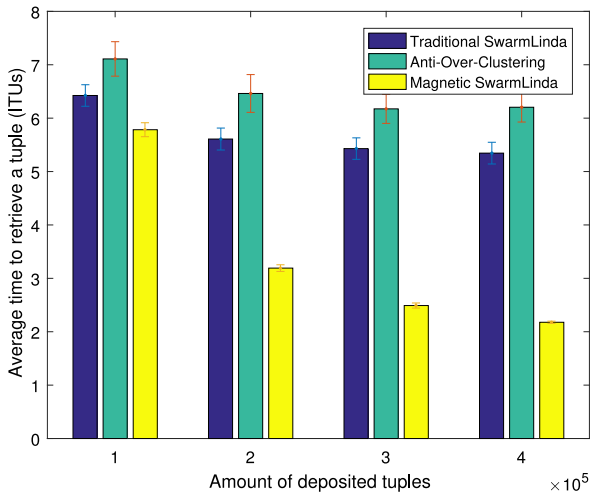

Fig. 8. Scenario 5 – Performance evaluation for different amounts of deposited tuples.

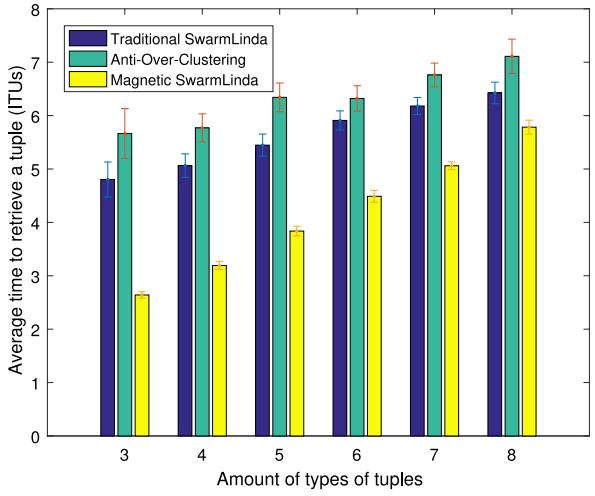

Fig. 9. Scenario 6 – Performance evaluation for different amounts of types of tuples.

number of tuples grew in comparison with *Magnetic SwarmLinda*, as shown in Fig. 8.

Finally, the results obtained for Scenario 6 (Fig. 9) demonstrate that *Magnetic SwarmLinda* is better adapted to scenarios where the amount of tuple types is smaller than the number of nodes that make up the Tuple Space. This happens because the lesser the amount of possible types of tuples, the smaller the number of magnetic clusters. In other words, each cluster can be formed by a larger number of nodes, which causes a considerable improvement in the overall performance. Although the other approaches also benefit from a smaller amount of types of tuples, they do that in a less significant way.

Concerning the round-trip delay of a *template-ant* as defined in Section 4.1, Fig. 10 demonstrates that Anti-Over-Clustering SwarmLinda shows a worse performance than the other two approaches even when the processing time is not considered. This is due to the fact that Anti-Over-Clustering scatters clusters of similar tuples over the Tuple Space, while *Magnetic SwarmLinda* maintains "clusters of clusters" (a.k.a. *magnetic clusters*).

For SwarmLinda with Anti-Over-Clustering, when a *template-ant* is in a cluster formed by tuples similar to its own template does not find a compatible tuple, it is penalized. This happens because the ant has to keep exploring the Tuple Space in order to find another cluster of matching tuples to eventually retrieve a compatible one.

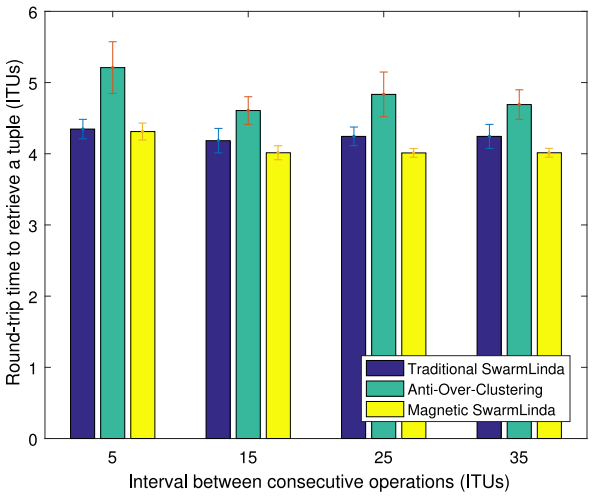

Fig. 10. The round-trip delay to retrieve a tuple.

The Traditional SwarmLinda and the proposed Magnetic approach do not suffer from this clustering dispersion problem. Actually, in the traditional approach, similar clusters are not even formed and nodes become overloaded. In the Magnetic approach, on the other hand, the over-clustering is avoided by the creation of magnetic clusters composed of nodes containing similar clusters of tuples. Thus, the performance impact when an ant does not find a compatible tuple in the first cluster visited is small since there are similar clusters nearby.

5. Conclusions

Tuple Spaces are an attractive alternative for building parallel and distributed systems as they provide a simple, elegant and flexible inter-process communication model. In this work, we introduced *Magnetic SwarmLinda* which combines swarm intelligence with virtual magnetic fields, to improve the scalability of Tuple Spaces by both avoiding overloaded nodes and by decreasing the average delay for executing tuple retrieval operations as it prevents the excessive dispersion of similar tuples. *Magnetic SwarmLinda* was implemented and results show that it outperforms other strategies, except in scenarios where similar tuples are very rare in the Tuple Space. This situation is nevertheless uncommon, as typical applications do not search for very specific (i.e., rare) tuples of which all the fields are known beforehand. In general, a search template includes one or more "blank" fields with only their types specified which will match multiple tuples. On the other hand, the gains of *Magnetic SwarmLinda* were always expressive as the amount of similar tuples grew, and its behavior remained stable regardless of the network density. In particular, important performance improvements were observed in scenarios where the number of nodes is greater than the number of types of tuples.

By improving the scalability of Tuple Spaces, *Magnetic SwarmLinda* may open new application fields of the paradigm of Distributed Shared Memory for the effective construction of very large distributed systems. There are multiple types of applications that can benefit from the temporal and spatial uncoupling provided by Tuple Spaces, in particular those involving Big Data, including the context of IoT. Future work also includes the development of a dynamic mechanism to determine the value of the magnetic restriction $M_c(i)$. Furthermore, the investigation of different similarity functions can further improve the performance of the system, which is specially relevant for applications with rare tuples since by changing the similarity function, a rare tuple may become ordinary.

References

[1] A. Atkinson, Tupleware: A distributed tuple space for cluster computing, in: Parallel and Distributed Computing, Applications and Technologies, 2008, PDCAT 2008, Ninth International Conference on, IEEE, 2008, pp. 121–126.

[2] A.N. Banks, R.B. Srygley, Orientation by magnetic field in leaf-cutter ants, Atta Colombica (Hymenoptera: Formicidae), Ethology 109 (10) (2003) 835–846.

[3] V. Buravlev, R. De Nicola, C.A. Mezzina, Tuple spaces implementations and their efficiency, in: International Conference on Coordination Languages and Models, Springer, 2016, pp. 51–66.

[4] M. Casadei, R. Menezes, R. Tolksdorf, M. Viroli, On the problem of over-clustering in tuple-based coordination systems, in: Self-Adaptive and Self-Organizing Systems, 2007, SASO'07, First International Conference on, IEEE, 2007a, pp. 303–306.

[5] M. Casadei, R. Menezes, M. Viroli, R. Tolksdorf, Self-organized over-clustering avoidance in tuple-space systems, in: 2007 IEEE Congress on Evolutionary Computation, IEEE, 2007b, pp. 1408–1415.

[6] D. Gelernter, Generative communication in Linda, ACM Trans. Program. Lang. Syst. 7 (1) (1985) 80–112.

[7] D. Gelernter, Multiple tuple spaces in linda, in: International Conference on Parallel Architectures and Languages Europe, Springer, 1989, pp. 20–27.

[8] D. Gelernter, A.J. Bernstein, Distributed communication via global buffer, in: Proceedings of the first ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, ACM, 1982, pp. 10–18.

[9] A. Hagberg, D. Schult, P. Swart, Networkx, 2013, http://networkx.github.io/index.html.

[10] H. Hari, Tuple Space in the Cloud (Master's thesis), Uppsala University, Department of Information Technology, 2012.

[11] Y. Jiang, G. Xue, Z. Jia, J. You, Dtuples: A distributed hash table based tuple space service for distributed coordination, in: Grid and Cooperative Computing, 2006, GCC 2006, Fifth International Conference, IEEE, 2006, pp. 101–106.

[12] M.E. Maia, R. Andrade, W. Viana, Towards a component infrastructure for cyber-physical systems, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing, ACM, 2016, pp. 626–628.

[13] R. Menezes, R. Tolksdorf, A new approach to scalable linda-systems based on swarms, in: Proceedings of the 2003 ACM Symposium on Applied computing, ACM, 2003, pp. 375–379.

[14] L.A. de Paula Lima Jr., A. Calsavara, Autonomic application-level message delivery using virtual magnetic fields, J. Netw. Syst. Manage. 18 (2010) 97–116.

[15] G.P. Picco, A.L. Murphy, G.-C. Roman, LIME: Linda meets mobility, in: Proceedings of the 21st International Conference on Software Engineering, ACM, 1999, pp. 368–377.

[16] A. Rowstron, WCL: A co-ordination language for geographically distributed agents, World Wide Web 1 (3) (1998) 167–179.

[17] T. SimPy, SimPy Overview, 2016, https://simpy.readthedocs.io/en/latest/ (Accessed 12 October 2016).

[18] R. Tolksdorf, R. Menezes, Using swarm intelligence in linda systems, in: Proc. 4th Int. Workshop Engineering Societies in the Agents World, LNCS 3071, 2004, pp. 49–65.

[19] E. Wajnberg, D. Acosta-Avalos, O.C. Alves, J.F. de Oliveira, R.B. Srygley, D.M. Esquivel, Magnetoreception in eusocial insects: an update, J. R. Soc. Interface (2010).

[20] J. Waldo, et al., Javaspace Specification-Revision 0.4, Technical report, Sun Microsystems, JavaSoft Lab, 1997.

[21] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world'networks, Nature 393 (6684) (1998) 440–442.

Henrique Duarte Lima has obtained his graduate degree in computer engineering at the Pontifical Catholic University of Paraná-PUCPR in 2013 and a Master degree in Computer Science at the same university in 2016. In the industry, he has experience with research and development (R&D) in the following subjects: cloud computing, Long-Term Evolution (LTE) Networks and distributed computing. His current main subjects of interest include: Swarm Computing, Speculative Parallelism and Internet of Things (IoT).

Dr. Luiz A. P. Lima Jr. is currently a full professor of computer science at the Pontifical Catholic University of Paraná-PUCPR (Brazil), leader of the Research Group on Distributed Systems. He graduated in Computer Science from the University of são Paulo-USP (Brazil) in 1991, got his master degree from State University of Campinas- UNICAMP (Brazil) in 1994 and his Ph.D. from the Université D'Évry- Val D'Essonne/National Institute of Telecommunications-INT (France) in 1998. He has since supervised several research projects and has published many papers in the areas of distributed and parallel algorithms, distributed systems and middleware, data replication, virtual magnetic fields, swarm computing, mobile computing and disruption-tolerant networks.

Alcides Calsavara received the B.Sc. in computer science from the University of Campinas, Brazil, and the Ph.D. in computer science from the University of Newcastle upon Tyne, UK. He is a Research Fellow with the Graduate Program in Informatics, Pontifical Catholic University of Paraná, Brazil., where he is a member of the Distributed Systems Group. His research interests include load balancing, mobility, middlewares, IoT, and smart cities.

Henri F. Eberspächer received the B.Sc. degree in Computer Engineering from PUCPR (1994), the M.Sc. degree in Industrial Informatics from UTFPR (1998), both in Brazil and the Ph.D. degree in Computer Science from Université Montpellier II/LIRMM – Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier – France (2007). He is full professor at the Computer Engineering Department of the Pontifical Catholic University of Paraná. His current research interests include mobile platforms, pervasive computing and Role-Based Collaboration.

Ricardo C. Nabhen is professor of electrical engineering at the Pontifical Catholic University of Paraná-PUCPR (Brazil) since 1991. He graduated in Electrical Engineering from the Federal University of Itajuba-UNIFEI (Brazil) in 1990, got his master degree in CS from the Pontifical Catholic University of Paraná-PUCPR (Brazil) in 2003 and his Ph.D. in CS from the Pierre et Marie Curie University - Paris 6 (France) in 2009 and from the Pontifical Catholic University of Paraná -PUCPR (Brazil) in 2010. He is currently professor and head of the Electrical Engineering undergraduate course and leader of the R&D Innovation Center in Computer Networks and Cloud Systems. He has since supervised several research projects in cooperation with the industry in the areas of mobile systems and cloud computing.

Elias P. Duarte Jr. is a Full Professor at Federal University of Paraná, Brazil. Research interests include Dependability, Computer Networks and Distributed Systems. He has published nearly 200 peer-reviewer papers, chaired more than 20 conferences and workshops, and is Associate Editor of the IEEE Transactions on Dependable and Secure Computing. He chaired the Special Interest Group on Fault Tolerant Computing of the Brazilian Computing Society (2005–2007); the Graduate Program in Computer Science of UFPR (2006–2008); and the Brazilian National Laboratory on Computer Networks (2012–2016). He is a member of the Brazilian Computing Society and a Senior Member of the IEEE.