Self-Organized Over-Clustering Avoidance in Tuple-Space Systems

Matteo Casadei, Ronaldo Menezes, Mirko Viroli and Robert Tolksdorf

Abstract—When it comes to communication performance, open distributed tuple-space systems depend heavily on the proximity of tuples to processes. Researchers have proposed many approaches for storing tuples in a way that processes benefit from the organization of tuples. Although some progress has been made, most of the proposed solutions fail to address the reverse problem: if most tuples are kept near the processes, the system’s robustness is affected; the over-clustering of tuples in particular nodes creates a dependence to that particular node. Hence, we have a dichotomy where it is important to organize tuples in clusters near the processes, but it is equally important to avoid over-clustering. The ideal is to have a balance where tuples are clustered but not totally concentrated in very few tuple spaces (eg. one or two). This paper presents a self-organized solution to the tuple distribution problem, in which the possibility of over-clustering is considered.

I. INTRODUCTION

LINDA is a coordination language based on a shared space of tuples that is manipulated by a minimal set of operations [1]. Its main advantage is the uncoupling between the coordinated entities—they interact in an asynchronous and anonymous fashion. The use of self-organized approaches in LINDA systems is becoming quite common. Approaches such as the one introduced by Mamei et al. [2] and by Menezes and Tolksdorf [3], [4] have started a trend in which solutions based on self-organization are used to ameliorate problems such as scalability and fault tolerance. The characteristics of self-organization such as local communications and lightness of agents, make it an ideal choice in the implementation of large scale systems.

In the specific case of tuple-space systems, the distribution of tuples in a network of tuple spaces is likely to be the single most important task affecting the scalability of a system. This problem has been addressed in several implementations of LINDA [5], [6], [7], [8]. For instance, Lime [8] features processes endowed with local tuple spaces where all the storing takes place. This is contrasted with standard approaches, which base tuple organization on the use of hashing of tuples to locations (tuple spaces). This approach was first championed by Obreiter and Gräf [9] who argue in favor of organizing tuples in servers based on the structure (templates) of tuples. Even though the use of hashing yields very efficient solutions, they are inappropriate to large distributed applications, because adaptation of hashing to dynamic scenarios is not trivial. While solutions such as Lime’s does improve on the standard case, it does not consider that the mobility of data is orthogonal to the mobility of processes. Menezes and Tolksdorf [3], [4] proposed a new direction for solving this problem. They argued that approaches inspired by Swarm Intelligence [10], [11] can be used to improve tuple organization and tuple retrieval in LINDA systems. Their approach is called SwarmLinda [3], [4] and argues that the placement of a tuple should be stochastically decided according to similarity between the tuple being stored and the others already stored in the tuple spaces.

Independently of the solution used, it is quite easy to disregard a problem resulting from tuple organization: over-clustering. While trying to organize tuples according to certain criteria, one can easily forget that the over-clustering of these tuples in one location may also be undesired. We characterize a situation as over-clustering when the tuple organization strategy forces all or most of the tuples (of a particular template) to be located in one tuple space. For this reason, over-clustering can be a threat to system integrity in open distributed tuple-space systems—if a node aggregating tuples fails, the whole system operation is compromised, for tuples of that particular template become unavailable in the system.

The tuple-hashing approach normally causes over-clustering, since all tuples of a template are placed in the same location. However, even self-organized solutions such as the ones proposed in SwarmLinda may yield [3] over-clustering as we discuss later in this paper. To overcome this problem we propose a mechanism where tuple organization takes into consideration the possibility of over-clustering. In essence, when tuples of the same template start to form a large cluster, the cluster itself stops being an attractor and it becomes a detractor. It is important to note that one has to achieve a balance between clustering and distribution of tuples. Clustering is desirable but not excessive clustering as it brings more problems than benefits.

This paper is organized as follows. In Section II we discuss the importance of tuple organization in SwarmLinda which continues into Section III where the distribution mechanism used in SwarmLinda is described in detail including an explanation about system entropy. The mechanism described for tuple distribution may yield over-clustering, hence we introduce in Section IV a strategy to avoid over-clustering. Section V then introduces some changes to SwarmLinda to make the process of over-clustering avoidance more efficient. The experiments described in Section VI confirm the efficiency of the proposed approach. Finally, we conclude the
paper in Section VII with some discussion on the open issues related to this approach.

II. TUPLE ORGANIZATION IN SWARMLINDA

The tuple organization process (or tuple sorting) is one of the most important tasks to be performed in the optimization of coordination systems. It relates primarily to the out primitive, since this is LINDA's way of allowing processes to store information. In Swarmlinda, the goal is to have similar tuples being placed closer to each other, forming a cluster on the basis of their template. To this end, Swarmlinda adopts several adaptations of algorithms taken from the abstraction of natural multi-agent systems [12], [10], [11]. Recently, new models taken from biology have been studied in computer science [13]. In particular, these models are characterized by actors that sacrifice individual goals for the benefit of the collective.

Swarmlinda's way to achieve tuple organization adopts ant's brood sorting [14] and is based on simple decisions taken by out-ants. These out-ants are processes inspired from Swarm Intelligence that roam on the terrain (network of tuple spaces) and decide at each hop of their roaming in the network if the tuple (food) should be stored. The decision is made stochastically, but biased by the amount of similar tuples around the ant's current location.

For the above to work well, there should be assurances that the tuple will eventually be stored. This is achieved by having an aging mechanism associated with the out-ant. For every unsuccessful step the out-ant takes, the probability of dropping the carried food-tuple in the next step increases and it is guaranteed to eventually reach 1 (one).

The similarity function is another important mechanism. Since having a monotonic scheme for the similarity of two tuples may be too restrictive in a real system, we want to have a function that says how similar the two tuples are. Note that the issue of goodness of a similarity function (how precisely it conveys the idea of similarity) is orthogonal to the problem we discuss here. Given a similarity function the tuples would self-organize accordingly. In this paper we show experiments where the tuples are all different, meaning that the similarity is monotonic.

III. SWARMLINDA DISTRIBUTION MECHANISM

The Swarmlinda distribution mechanism is designed and simulated by using an executable specification based on the stochastic simulation framework defined in [15]. Using this specification, we executed many experiments on the network of 14 tuple spaces shown in Figure 1—a scale-free topology [16].

We think of tuples as terms of the form $N(c_1, c_2, ..., c_n)$, where $N$ is called the tuple name and $c_1, c_2, ..., c_n$ are the tuple arguments, namely, constant values like literals or numbers. Then, we consider a similarity function $\delta$ that takes two tuples and compares them, yielding a real value in between 0 and 1. The higher this value the more similar the two tuples are, with 1 meaning the tuples are identical and 0 that they are completely different.

The specific implementation of this similarity function is of no interest in the context of this paper—different application contexts can call for different similarity functions. A study of the appropriate similarity function for an application needs to be done by considering many issues such as number of tuple templates, topology of the network, and application domain.

The Swarmlinda solution to the distribution problem is composed of a decision phase that decides whether to store a tuple $tu$ in the current tuple space, and a movement phase that chooses the tuple space for the next hop, if the decision taken in the previous phase was not to store $tu$ in the current tuple space.

A. Decision Phase

During the decision phase, the out-ant primitive has to decide whether to store the carried tuple $tu$ (food) in the current tuple space $TS$ or not. This phase involves the calculation of the concentration $F$ of tuples having a template similar to $tu$, using the similarity function, and the calculation of the probability $P_D$ to drop $tu$ in the current tuple space. The concentration $F$ is calculated by considering the similarity of $tu$ with all the other tuples $t$ in the tuple space, given by:

$$F = \sum_{t \in TS} \delta(tu, t)$$

Note that $0 \leq F \leq Q$, where $Q$ is the total number of tuples within the current tuple space (for all templates). According to the original brood sorting algorithm used in Swarmlinda and inspired by Deneubourg et al. proposal [14], the probability $P_D$ to drop $tu$ in the current tuple space is given by:

$$P_D = \left(\frac{F}{F + K}\right)^2$$

with $0 \leq P_D \leq 1$. Differing from the original idea in brood sorting, here the value of $K$ is not a constant; $K$ represents
the number of steps remaining for the \texttt{out\{tu\}} primitive, namely, the number of tuple spaces that an \texttt{out-ant} can visit. When an \texttt{out} operation is initially requested on a tuple space, the value of \( K \) is set to \textit{Step}, that is, the maximum number of tuple spaces each \texttt{out-ant} can visit.

Each time a new tuple space is visited by an \texttt{out-ant} without storing the carried tuple \( tu \), \( K \) is decreased by 1. When \( K \) reaches 0, \( P_0 \) becomes 1 and \( tu \) is automatically stored in the current tuple space, independently of the value of \( F \). In other words, \( K \) implements an \textit{aging mechanism} for an \texttt{out-tuple}.

\textbf{B. Movement Phase}

The \textit{movement} phase occurs when the tuple carried by an \texttt{out-ant} is not stored in the current tuple space. The goal is to choose, from the neighborhood of the current tuple space, a good neighbor for the next hop of the \texttt{out-ant}. The best possible neighbor is the tuple space with the highest concentration \( F \) of tuples similar to the carried tuple \( tu \).

If we denote by \( TS_1, \ldots, TS_n \) the tuple spaces in the neighborhood of the current tuple space, \( F_j \) the concentration of tuples similar to \( tu \) in \( TS_j \) (obtained using Equation 1), then the probability \( P_j \) of having an \texttt{out-ant} move to neighbor \( TS_j \), is calculated as follows:

\[ P_j = \frac{F_j}{\sum_{k=1}^{n} F_k} \quad (3) \]

The higher the value of \( P_j \) for a neighbor \( TS_j \), the higher the probability for that neighbor to be chosen as the next hop of the \texttt{out-ant}. After a new tuple space is chosen, the whole process is repeated starting from the decision phase.

\textbf{C. Experimental Results}

Using the network shown in Figure 1, we simulated the storage of tuples of four different templates: \( a(X) \), \( b(X) \), \( c(X) \) and \( d(X) \). In particular, we simulated the insertion of 210 tuples for each template.

As a measure of the obtained degree of order in the network, we adopted the concept of \textit{spatial entropy}. Denoting by \( q_{ij} \) the amount of tuples matching template \( i \) within tuple space \( j \), \( n_j \) the total number of tuples within tuple space \( j \), and \( k \) the number of templates, the entropy associated with tuple template \( i \) within tuple space \( j \) is

\[ H_{ij} = \frac{q_{ij}}{n_j} \log_2 \frac{n_j}{q_{ij}} \quad (4) \]

and it is easy to notice that \( 0 \leq H_{ij} \leq \frac{1}{k} \log_2 k \). We want to express now the entropy associated with a single tuple space

\[ H_j = \frac{\sum_{i=1}^{k} H_{ij}}{\log_2 k} \quad (5) \]

where the division by \( \log_2 k \) is introduced in order to obtain \( 0 \leq H_j \leq 1 \). Moreover, a value of \( H_j \) equals to 1 corresponds to a situation of complete chaos in tuple space \( j \), since we have \( q_{ij} = \frac{n_j}{k} \) for \( 1 \leq i \leq k \). Oppositely, a value of \( H_j \) equals to 0 corresponds to a situation of complete order, since all the \( n_j \) tuples in tuple space \( j \) are of the same template. If we have \( t \) tuple spaces, then the spatial entropy of a network is

\[ H = \frac{1}{t} \sum_{j=1}^{t} H_j \quad (6) \]

where the division by \( t \) is used to normalize \( H \), so that \( 0 \leq H \leq 1 \). The lower the value of \( H \), the higher the degree of order in the network.

We performed series of 20 simulations, using each time different values of the \textit{Step} parameter. A simulation consists of the insertion of tuples in the network of tuple spaces—via \texttt{out} primitive—until there are no pending \texttt{out} to be executed in the network. After the execution of 20 simulations, we calculated the value of the \textit{spatial entropy} \( H \) of the network as the average of the single values of \( H \) resulting from each simulation; this value is called \textit{average spatial entropy} \( H_{\text{avg}} \).

The chart reported in Figure 2 shows the trend of \( H_{\text{avg}} \) using different \textit{Step} values. We can see that, the higher the value of \textit{Step}, the lower the value of \( H_{\text{avg}} \). Moreover, if we use a \textit{Step} value greater than 25, we achieve \( H_{\text{avg}} \approx 0.05 \), that is the lowest achievable value of \( H_{\text{avg}} \). Such a value means a quasi-complete clustering of the different tuple templates in the network, displaying the capability of the SwarmLinda distribution mechanism to achieve a quasi-perfect tuple distribution.

Furthermore, the charts reported in Figure 3 show the evolution of the formation of clusters in tuple space 10 and tuple space 13 with \textit{Step} = 30. Note that similar tuples are placed together forming clusters per tuple template. Again, even though clustering is a desirable feature, it is often source of the problems related to over-clustering and already described in Section I.

\textbf{IV. AVOIDING OVER-CLUSTERING IN SWARMLINDA}

The results presented in Section III clearly show that over-clustering can take place if the SwarmLinda algorithm is used

\[ Fig. 2. \text{ Trend of average spatial entropy } H_{\text{avg}} \text{ resulting from the simulation of the scale-free network shown in Figure 1.} \]
without care. In order to avoid over-clustering, we introduced the parameter Max-Size, that represents the maximum number of tuples allowed for each cluster. Then we exploit a generalized logistic function (a sigmoid function), so that the probability \( P_D \) can be normalized according to a sigmoid curve. More precisely, we used an instance of the generalized logistic function, characterized by the equation:

\[
P_D' = P_D - \left[ 0.01 + \left( \frac{P_D - 0.01}{1 + 0.5e^{-b(X-M)^2}} \right) \right]
\]

where \( P_D \) is the drop probability calculated by using Equation 2, \( b \) is the slope of the curve and \( m \) is the value of the \( X \) variable at which we observe the maximum value of the curve derivative. Equation 7 defines a complemented generalized logistic function, since we want to obtain the maximum value of \( P_D' \) when \( X \) has a small value. Indeed, the value assigned to \( X \) is the concentration \( F \) calculated considering the current tuple to be stored. The value of \( m \) depends on the total number of tuples per template expected in the network, while the slope value \( b \) depends on the values chosen for Max-Size: the lower the value of Max-Size, the higher the value of \( b \). Figure 4 shows the function used to obtain a value of \( b \) given a value of Max-Size.

According to Equation 7, as the number of tuples of a given template (the concentration \( F \)) approximates to Max-Size, the drop probability \( P_D' \) degenerates towards 0 (zero). More precisely, the \( P_D \) value calculated by Equation 2 is used as the maximum value returned by our generalized logistic function when \( F = 0 \). Then, depending on the concentration \( F \) calculated by Equation 1, the new value \( P_D' \) of drop probability is returned by adopting our generalized logistic function—again, a simple normalization based on the current level of clustering.

In our experiments Max-Size is set to 100: this choice was based on the number of tuples stored in the network, and on the size of the network itself. Figure 5 presents the complemented generalized logistic function obtained with Max-Size = 100 and \( P_D = 0.9 \). In particular, the value of \( b \) is obtained by using the function shown in Figure 4 with Max-Size = 100. Then applying the obtained value of \( b \) to Equation 7, we can generate the curve shown in Figure 5. According to Equation 7, this curve features a lower asymptote equals to 0.01, and an upper asymptote equals to \( P_D \). In our future work, we intend to devise a solution based on a dynamic Max-Size parameter. In particular, we would like Max-Size to be a function of the clustering status, the size of the network, or perhaps of the current entropy of a tuple space (its level of organization).

Adopting this anti-over-clustering strategy, we performed the same simulations described in Section III. The results are shown in Figure 6. Comparing these results with the ones reported in Figure 3, we can see that the improvement, in terms of over-clustering avoidance, is significant. Indeed, both tuple spaces 10 and 13 are now characterized by clusters of smaller size than the case with no anti-over-clustering strategy.

Choosing template \( c(x) \) as reference, Figure 7 compares the concentration of tuples of template \( c(X) \) in the network with and without over-clustering. While in the case without anti-over-clustering we have a heavy concentration of \( c(X) \) tuples on one tuple space, now the same amount of tuples is divided on two tuple spaces. More importantly, we do not loose the clustering ability—the tuples are clustered in a group of close nodes rather than in individual nodes far from each other.

V. ENHANCED ANTI-OVER-CLUSTERING STRATEGY

Although the approach described in Section IV provides already a good solution to avoid over-clustering, some improvements are needed. In particular, we need to modify the original SwarmLinda distribution mechanism presented in Section III, adapting its behavior to the anti-over-clustering
strategy described in Section IV. Indeed, in Section IV only the decision phase of the SwarmLinda distribution mechanism is adapted to cope with over-clustering, whereas the movement phase is left unchanged. As a major consequence, when a tuple is not stored in the current tuple space, the choice of a good neighbor does not take into account over-clustering, making it possible to choose an over-clustered neighbor for the next hop. Since we wanted to have a distribution mechanism completely coherent with the anti-over-clustering strategy presented in Section IV, we developed a new movement phase by modifying Equation 3.

More precisely, the new probability $P_j$ of having an out-ant move to neighbor $TS_j$, is:

$$P_j = \frac{P_j'}{\sum_{k=1}^{n} P_k'}$$

where $P_j'$ is obtained by Equation 7. Fundamentally, the new calculation for $P_j$ considers how good $P_j'$ is when compared to all the other probabilities of the neighbors. In other words, it takes $P_j'$ as a relative value instead of an absolute one.

Again, the slope value $b$ used to calculate $P_j'$ depends on the chosen Max-Size value. The division by $\sum_{k=1}^{n} P_k'$ shown in Equation 8 is a normalization that makes it possible to have:

$$\sum_{j=1}^{n} P_j = 1$$

Adopting this new movement phase, we obtained a completely coherent SwarmLinda distribution mechanism characterized by an enhanced anti-over-clustering strategy. Note that the main advantage of this approach is to consider over-clustering even in the movement phase; though, since we are using a self-organized approach, we do not want to completely avoid the choice of an over-clustered tuple space, but rather, to make it less likely than other tuple spaces.

The next section provides a description of the results.
achieved by applying the enhanced anti-over-clustering strategy on the scale-free network shown in Figure 1.

VI. RESULT EVALUATION OF THE ENHANCED ANTI-OVER-CLUSTERING STRATEGY

We performed a set of three simulations adopting the enhanced anti-over-clustering strategy on the scale-free network reported in Figure 1. Since we were mainly interested in the observation of the tuple distribution for different Max-Size values, we decided to use only tuple template \( c(X) \). More precisely, for each simulation, we simulated the insertion of 840 tuples—60 per tuple space. Each simulation was run adopting a different Max-Size value: in particular, we considered Max-Size equals to 100, 400, and 840. Every execution was performed using \( \text{Step} = 30 \). The choice of \( \text{Step} = 30 \) corresponds to a very critical condition for the enhanced anti-over-clustering strategy, since such a value, if we do not use any anti-over-clustering strategy, guarantees a situation of quasi-complete clustering, as shown in Figure 2. Moreover, this choice was driven by the necessity of testing our enhanced anti-over-clustering strategy in the worst case.

Figure 8 shows the sigmoid functions corresponding to the Max-Size values chosen for the simulations. It is easy to see that the higher the Max-Size value, the lower the value of \( b \). Given a value of \( F \), a lower value of \( b \) generates a higher value of \( P_D' \): this corresponds to a higher probability to store a tuple in the current tuple space. In other words, lower values of \( b \) allow the formation of larger clusters. In particular, the cluster size is directly related to the chosen Max-Size value: indeed, fixing Max-Size, the probability \( P_D' \) quickly decreases as the concentration \( F \) becomes close to the fixed Max-Size value.

The results obtained by running the three simulations are shown and compared in Figure 9. The Figure makes it evident the influence of Max-Size on the cluster size. Indeed, with Max-Size = 100 we obtained clusters characterized by an average size equals to 100 tuples, the simulation with Max-Size = 400 led to the formation of two clusters with an approximate average size equals to 400 tuples. Moreover, with Max-Size = 840 we obtained a quasi-perfect clustering, since the Max-Size value is equal to the number of tuples inserted in the network: as a consequence, even though this situation features over-clustering, we demonstrated that Max-Size can be used to tune the system behavior in a custom way. These results show that we can use Max-Size to influence the tuple distribution in a self-organized fashion.

Figure (a), (b) and (c) provide an alternative view of the tuple distribution resulting from the different Max-Size values. More precisely, tuple distribution is shown referring to network topology. It is easy to see that every value of Max-Size guarantees the formation of close clusters, maintaining in any case a good level of organization of tuple distribution. We can also recognize that more connected tuple spaces have a higher probability to get stored a high number of tuples.
VII. CONCLUSION

In this paper we developed a solution to the over-clustering problem that can arise from solutions for tuple distribution such as the one proposed in SwarmLinda. This work was initially driven by the consideration that over-clustering may occur even though we adopt self-organized solutions based on newer approaches. Using the proposed solution, we executed some simulations considering the case without anti-over-clustering and the case with anti-over-clustering. In particular, this first solution to over-clustering was developed by considering the effect of over-clustering in the probability to store a tuple in a node.

Then we improved the first anti-over-clustering strategy, making the entire SwarmLinda distribution mechanism coherent with the previously developed anti-over-clustering strategy. In this improved solution the effect of over-clustering is considered even in the movement phase, that occurs when the considered tuple is not stored in the current tuple space. This enhanced anti-over-clustering strategy was tested with different Max-Size values, showing how Max-Size affects the tuple distribution in the network.

There are many improvements that need to be performed in this approach but, most importantly, we need to devise a truly self-organized solution able to dynamically adapt the Max-Size value. The value of this parameter drives the clustering behavior, as one can see in Figure 10. However the particular value may be inadequate as the system changes. For instance, a system with one tuple space containing 10 tuples may be considered over-clustered if 10 tuples are all the tuples in the system. What one would like is for the value to change as some of the characteristics change, such as the size of the network (number of nodes), the reliability of the nodes, the...
total tuple-space number in the system, and the number of tuple kinds in the system. However, the challenge is due to the fact that our self-organized approach should not be based on global parameters that are costly to maintain.

Last, we intend to devise metrics to evaluate the effect of clustering (with and without the approach to anti-over-clustering) in the survivability of SwarmLinda. Fault-tolerance and survivability of open Linda systems is rarely tackled. We believe our approach improves the fault-tolerance of SwarmLinda and should make it a good choice for faulty systems. We are currently working on a framework to inject faults in the system and evaluate SwarmLinda’s performance.

REFERENCES