On the Performance of Swarm-based Tuple Organization in Linda Systems

Daniel Graff, Ronaldo Menezes, Robert Tolksdorf

Abstract—Coordination systems have been gaining popularity since the early 80s with the introduction of the LINDA coordination model. Soon after its introduction researchers and practitioners alike started to realize that coordination is ubiquitous to any distributed systems but unfortunately partially responsible for the inefficiency found in these systems—coordination deals with costly issues such as process communication and synchronization. In the beginning of this decade, researchers looked for alternatives for implementing more efficient means of coordination; they turned to Swarm Intelligence; the first of these approaches was called SWARMINDA. Performance of LINDA-based coordination systems is directly related to the issue of system entropy (tuple organization). SWARMINDA approaches for tuple organization are investigated in this paper using a simulator. After a careful study on the performance of SWARMINDA, we introduce modifications on the algorithm so as to achieve better entropic levels.

I. INTRODUCTION

SINCE the early days of coordination with the introduction of the LINDA coordination model [1], it became clear that coordination is an issue ubiquitous to all distributed applications. The mere distribution of resources on a network coupled with the use of concurrency makes coordination condition sine qua non in the organization of activities by these concurrent processes. Not surprisingly, LINDA has been used as the basis for coordination in many applications ranging from parallel computing ([2], [3]) to mobile systems ([4], [5]) to peer-to-peer systems [6].

Anyone would agree that the complexity of computer applications around us is increasing. While a couple of decades ago we would be happy to have applications helping us do our taxes, today’s applications are helping us understand the universe (e.g., Grid computing being use in Physics experiments at CERN). In these complex, large-scale applications, standard methods of coordination become inadequate due to their inability to scale to such sizes. It was with this in mind that SWARMINDA [7] was proposed. SWARMINDA is a coordination model constructed entirely on the notion of swarm systems. Since its conception we discussed how it could be implemented [8], and demonstrated that its implementation exhibit properties of self-organization [9].

This paper sets to evaluate the ability of SWARMINDA to organize tuples efficiently, meaning its capability to achieve good entropy. The organization of tuples in LINDA systems is the most important factor of performance since defines the entropy of the system. Good entropy (~ 0) means that most tuples of the same type are kept clustered, bad entropy (~ 1) signifies a chaotic pattern where tuples are stored without any clear organization. In [10], we discussed in length the importance of tuple organization and the issue of clustering vs. over-clustering. Hence, given the importance of the issue, we evaluate in this paper the approaches proposed in the original SWARMINDA and suggest improvements whenever necessary.

The paper is organized as follows. In Section II we discuss the general idea of the use of swarms and self-organized approaches in LINDA systems. It follows in Section III with a critique of the original approaches (metrics) for calculating performance of SWARMINDA proposed in earlier works, and the introduction of modified metrics. In Section IV we describe the simulator implemented in NetLogo and its main features. In Section V we present the main results found in this work, in particular the demonstration of the self-organized characteristics of three algorithms of SWARMINDA. Finally, we discuss our future work plans and conclude this work in Sections VI and VII respectively.

II. SWARM-BASED COORDINATION WITH LINDA

Before we discuss coordination based on Swarm approaches, it is important that we write a few words about the coordination model that is the basis for this paper. LINDA is a coordination model proposed by Gelernter [1] and later extended by the same author [11] making it more suitable to distributed environments. LINDA advocates that processes communicate using generative communication; processes store information (tuples) in associative memories (tuple spaces) and these tuples may be retrieved by other processes using content addressing. Since communication takes place via these tuple spaces, messages being exchanged are similar to data: both are represented as tuples in tuple spaces. The organization of these tuples is the main driving force for performance in LINDA.

Swarm-based coordination is the idea that all coordination activities are implemented using swarm-inspired approaches. The idea started after some researchers looked for LINDA as the basis for coordination in large scale applications. The problem is that in large scale scenarios, most LINDA systems do not perform well because they do not scale with the number of processes and network locations. SWARMINDA [7] was
proposed with one goal in mind: scalability. Despite this clear goal, a true achievement of scalability has many facets:

- Scalable systems require some level of fault-tolerance. When dealing with large-scale systems, it is unreasonable to assume a failure-free system.
- Scalable systems require extreme decentralization and autonomy of objects, meaning that any level of centralization could be a hurdle.
- Scalable systems require data distribution to be effective since access to unorganized data could cause acceptable overhead levels. In general, data should be kept organized by data type because processes (in LINDA) handle few data types at a time.
- Scalable systems require proximity. In order to avoid overhead, data not only should be organized by type, they should be kept close to the processes that require them. The consequence of this is that data should be mobile, moving from location to location with the goal of being kept close to data of the same kind and processes requiring them.

SWARM/LINDA is an attempt to address all the issues above. In particular, fault tolerance is achieved via a mechanism of good tuple distribution with over-clustering avoidance [10]; decentralization is achieved using biologically-inspired approaches in the process of tuple storage, organization and tuple retrieval. In the sense of these characteristics, SWARM/LINDA is a true self-organized model [9] meaning that it clearly shows characteristics of self-organized systems as proposed by Wolf and Holvoet [12].

Increase in Order: SWARM/LINDA has been shown to decrease the entropy of the system with regards to tuple organization within the system. Casadei et al. [9] introduces a mechanism to calculate the entropy of SWARM/LINDA and shows that it tends to decrease as the system executes. In this paper we modify this entropy calculation to cope with aspects that were overlooked by Casadei et al.; this paper confirms the tendency to order found earlier.

Autonomy: Increase in order is necessary but not sufficient to characterize self-organization; the increase in order cannot be due to a centralized controller or decision maker. The system needs to organize without interference from the external world. In SWARM/LINDA the lack of external intervention is not characterized by the execution of the LINDA primitives but rather by the fact that the process executing the primitives does not interfere on the organization of tuples in the system.

Adaptability/Robustness to Changes: Self-organized systems should be able to handle changes in the environment without the need for introducing changes in the behavior of the system. Here, changes in the network where tuples are stored do not affect the behavior of SWARM/LINDA.

Dynamic Nature: This relates to the characteristic that makes self-organized systems far from equilibrium meaning that the system does not reach an organization level that causes it to be static. The dynamism has to outline any organization so that the system is able to achieve adaptability. SWARM/LINDA contains tuple movement agents which are constantly trying to move tuples around. This dynamic aspect of SWARM/LINDA enables search for lower entropy levels even in the absence of new information (tuples) being stored and/or retrieved.

The main focus of this paper is on demonstrating the entropy behavior of SWARM/LINDA under the activities of three approaches that exist in SWARM/LINDA that are directly linked to tuple organization: tuple distribution, tuple movement and tuple retrieval (all three originally described in [7]).

A. Tuple Distribution

Tuple distribution is implemented via a mechanism inspired by the behavior of ants in nature [13]. When tuples are stored in some tuple space, the process responsible for finding a location to store the tuple (a tuple-ant) leaves some information on the environment to indicate the choices taken by the agent storing the tuple. As in ant colony optimization, this information is used by subsequent tuple-ants in their decision process.

We believe that the use of self-organization can overcome the major current limitation of LINDA systems: scalability. Known approaches such as intermediate distribution or tuple-rendezvous schemes do not scale since they take a global view on the space. With self-organization decisions are taken locally and independent of each other, therefore higher scalability is achieved.

The decision to store a tuple of a kind \(k\) is influenced by the total amount of tuples of kind \(k\) in a defined neighborhood of the tuple-ant as well as the age of the tuple ant. The probability to store a tuple at location \(\ell\) is directly proportional to the amount of tuples in that location but inversely proportional to the age of the tuple ants. If an ant does not store a tuple, it moves to a (more attractive) neighbor node but ages with the movement. The older the tuple-ant the more likely it is that it will store the tuple.

B. Tuple Movement

The dynamism of today’s applications means that processes tend to be mobile. Once tuples are stored, SWARM/LINDA does not understand them as fixed to that location. This approach for tuple movement is inspired by the cemetery organization behavior of some ant species [13].

In SWARM/LINDA, we have tuple-ants that are constantly trying to improve the entropy of the system (i.e. increase the organization). As processes move to different locations, tuples may get stored in locations where they are not well clustered based on their kind. Tuple-ants responsible for tuple movement constantly pick up tuples, carry them around in the system, and drop them in more suitable locations. A tuple-ant picks up a tuple with a probability that is inversely proportional to the clustering level of tuples of that kind where the tuple-ant is located. This means that the higher the entropy of that location the more likely the tuple-ant will pick up a tuple. Once it picks it up, the ant moves and drops the tuple with a probability...
directly proportional to the clustering level of tuples of that location in the new location where the tuple-ant moved to. Note that tuple-ants responsible for tuple movement do not age and are always active in the SWARM-LINDA system.

C. Tuple Retrieval

Tuples in SWARM-LINDA are retrieved based on the format of the tuple: content addressing. The approach for tuple retrieval of tuples is very similar to the process of tuple storage except for the fact that now we have template-ants; ants that carry the format of a desired tuple.

These template-ants move in the environment using information left on the environment itself from previous successful tuple retrievals and tuple storages. A template-ant carrying the format of a tuple of kind \( k \) will look for information in the environment that indicates where tuples of type \( k \) could be stored. Template-ants, also age and may fail to retrieve tuples meaning that the SWARM-LINDA most likely does not contain a tuple of that particular kind.

III. Metrics and Improved Algorithms

In this section we describe how we measure performance in SWARM-LINDA based on the entropic levels of the system. We follow with a modified version of how tuple-ants and template-ants behave in SWARM-LINDA wrt. storing tuples, moving tuples and retrieving tuples.

A. System Entropy

As already mentioned, we need a way for determining the level of order in the system. To evaluate the used algorithms and the system behavior itself we applied the concept of spatial entropy as introduced in [14]. The entropy was calculated by adding up the local entropies of each node (tuple space) and dividing it by the amount of nodes in the network. The entropy level ranges from 0 (for complete order) to 1 (for total chaos).

The amount of information objects (tuples) of the same kind in a node determines the entropy. Large amount of tuples of the same kind means low entropy values.

In a system with complete order, the information objects are distributed among the nodes forming cluster of similar objects. Depending on how this is calculated one may be looking at the entropy at a specific time but not take into consideration the actual distribution of tuples among the nodes. In general a good entropy calculation should show if the environment tends to organization or chaos. As proposed by Casadei et al. [14], even a low entropy value can pass the incorrect idea of organization (see Equation 1).

For instance, assume a network with 20 nodes with 200 tuples of 4 different kinds (50 of each kind). If we organize this network by placing all 200 tuples on one specific node (while keeping the remaining 19 ones empty), we will get according to Casadei et al.’s approach an entropy value of 0.05 which seems quite good. But at close inspection, one can see that this value is not realistic of the actual state because the idea of a good entropy should be that the network is ordered meaning that we find clusters among the nodes containing similar information. But in this case we will find one node containing all objects while the other ones are empty.

One could argue that this network is organized but in fact, the only reason we get this low value of the entropy is because the other nodes have entropy 0 (zero) since they are empty. So the chaos of the node containing all the tuples (which indeed it has) evaporates when one makes an average of entropies.

Again, note that the nodes with entropy 0 (zero) are only considered organized because they have no tuples. Considering this problem led us to the idea of calculating the weighted entropy by assigning a weighting factor to each local node entropy.

\[
H = \frac{\sum_{j=1}^{n} H_j}{L}
\]  
\[
H_{\text{weighted}} = \frac{\sum_{j=1}^{n} (m_j H_j)}{\sum_{j=1}^{n} m_j}
\]

Equation (1) shows the original formula for calculating the entropy as proposed in [14]. The idea is to sum the local entropies of the respective nodes and divide it by the number of nodes in the network indicated by \( L \). The approach is based on the theory that each node gets the same importance independent of the distribution of tuples. In contrast, Equation (2) postulates that each node gets a weighting value \( m_j \) that indicates the importance of that node in the network. The weighting factor is proportional to the local amount of tuples stored. Therefore \( m_j \) is the amount of tuples stored in location \( j \).

If we now return to the aforementioned scenario and re-calculate the entropy based on \( H_{\text{weighted}} \) in Equation 2, we will get an entropy value of 1.0 which indicates chaos in the networks. Compared to the previous value of 0.05 we can see that the proposed approach is more realistic. In this specific situation we also have a single point of failure since all tuples are stored in the same tuple space, thus another reason not to consider this network ideal reinforcing the claim that the entropic value given by Equation 1 is not realistic.

B. Tuple-Ants and Template-Ants Behavior

The tuple distribution is the basic mechanism used by executing the \texttt{out} primitive for storing objects in the network (this process is summarized in Algorithm 1). In the beginning the specified amount of ants, given by \texttt{num}, are instantiated in the system environment. They then enter each in a process of executing 3 steps:

Decision Phase: In the beginning each ant decides whether to drop the carried tuple denoted by \texttt{carriedTuple} at the current location \((TS)\). The probability of dropping it at the ants current node depends on the concentration of similar tuples contained in the tuple space on the current node (given by \texttt{conc}). Once the ant decides to drop the tuple it leaves pheromone—scent that can be sensed by other ants—at the current location which will then spread to the surrounding neighbors. The existence
of pheromone increases the attractiveness of a node. After dropping the tuple the work of the ant is done and it is removed from the simulation.

**Movement Phase:** In this step, the ants, after deciding not to drop the tuple in the previous step, have to select another good location based on the attractiveness of these locations which comes from pheromone information in these locations. The function $SelectNeighbor(TS)$ selects the most suitable tuple space neighbor of $TS$ and then the ant moves to that location. Note however that if no pheromone information is found around the ant’s current location, the ant will randomly choose a neighbor. This scenario is common in the case of an empty network but as soon as it got trained which will take just a few steps it is very likely that there will be at least a small amount of pheromone.

**Aging Phase:** Last, after moving to a new location, the ants always age and if the time-to-live factor ($ttl$) reaches 0 (zero) the ant drops the tuple independently of information around it. This step is important to guarantee that independently of the status the network the ant is guaranteed to eventually drop the tuple. If an ant reaches its maximum lifespan it is removed from the simulation ($Die$).

During the tuple distribution the ants should decide carefully at which location they drop the carried tuple in order to establish a high level of order in the network. Bad decisions result in an increase of the entropy and also degrades the probability, even when it is infinitesimal, for further ants to behave correctly. Of course there is always a specific amount of ants assigning their tuples to nodes that will result in an increase of the entropy, but the system is robust against this since most of the ants finish their task successfully thus establishing a high level of order.

We applied the same formula for calculating the drop probability of tuples as introduced in [14] and shown in Equation (3). $C$ is the amount of tuples at the current node which are similar to the one the ant carries. $K$ is equal to the $ttl$ value that indicated the number of steps the ant has left for accomplishing its task. It is easy to see that the drop probability depends on $C$ as well as $K$. Especially for small values of $C$ the $K$ value is controlling the drop probability. In that case the ant is supposed to move around for finding more appropriate tuple spaces. After some time the ant will give up and store the tuple anyway. For $C \gg K$, the $ttl$ value has minimum influence in the probability. The amount of similar tuples is so big that visiting ants tend to drop their tuple. Unfortunately, this also happens even when the tuple space is heterogeneous meaning that we have many different kinds of tuples stored at the same location. Thereby we get an increase of the entropy because it the Equation 3 incorrectly assumes homogeneity.

$$P_{orig} = \left( \frac{C_{orig} + K}{C_{orig}} \right)^2$$ (3)

**Algorithm 1:** This shows the process of implementing an $out$ command in the current simulator. Each ant makes a decision whether to drop a tuple in the current location ($DECIDE$), move to the next location location ($MOVE$), and gets older ($AGE$).

```
Algorithm 1

<table>
<thead>
<tr>
<th>OUT</th>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ants[] ← InstantiateAnts (num)</td>
<td></td>
</tr>
<tr>
<td>repeat</td>
<td></td>
</tr>
<tr>
<td>foreach ant ∈ ants[] do</td>
<td></td>
</tr>
<tr>
<td>DECIDE()</td>
<td></td>
</tr>
<tr>
<td>MOVE()</td>
<td></td>
</tr>
<tr>
<td>AGE()</td>
<td></td>
</tr>
<tr>
<td>until ttl = 0;</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DECIDE</th>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>simTuples ← CntSimilarTuples (TS, carriedTuple)</td>
<td></td>
</tr>
<tr>
<td>totalTuples ← CntTuples (TS)</td>
<td></td>
</tr>
<tr>
<td>conc ← CompConcentration (simTuples, totalTuples)</td>
<td></td>
</tr>
<tr>
<td>dropProb ← CompDropProbability (conc, ttl)</td>
<td></td>
</tr>
<tr>
<td>if Random (1) ≤ dropProb then</td>
<td></td>
</tr>
<tr>
<td>DropTuple (TS, carriedTuple)</td>
<td></td>
</tr>
<tr>
<td>DropPheromone ()</td>
<td></td>
</tr>
<tr>
<td>Die ()</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MOVE</th>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>nextNeighbor ← SelectNeighbor (TS)</td>
<td></td>
</tr>
<tr>
<td>MoveTo (nextNeighbor)</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>AGE</th>
<th>begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>ttl ← ttl - 1</td>
<td></td>
</tr>
<tr>
<td>if ttl = 0 then</td>
<td></td>
</tr>
<tr>
<td>DropTuple (TS, carriedTuple)</td>
<td></td>
</tr>
<tr>
<td>Die ()</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>
```

**Considering this problem led to the idea of calculating the actual concentration $C_{new}$ instead of using the amount of similar tuples.**

$$F_{sig}(x) = \frac{1}{1 + e^{-(20x-10)}}$$ (4)

$$C_{new} = F_{sig} \left( \frac{m_{ij}}{m_j} \right) m_{ij}$$ (5)

$$P_{new} = \left( \frac{C_{new}}{C_{new} + K} \right)^2$$ (6)

**Equation (6) shows the new drop probability. The basic modification is that the probability for dropping a tuple depends now on the amount of similar objects but also on the fraction of dissimilar ones as defined in Equation (5), where $m_{ij}$ denotes the set of tuples matching template $i$ within tuple space $j$, and $m_j$ denotes the total amount of tuples in $j$. The ratio expresses the fraction of similar tuples in contrast to dissimilar ones.** The applied sigmoid function given
in Equation (4) is used in order to reinforce resp. extenuate the ratio of \( \frac{m_{ij}}{m} \). The sigmoid function is normalized in \([0,1]\) since its input value \( x \) owns the same co-domain. While values being beneath 0.5 get compressed, values above 0.5 are forced to adopt higher values. This assures that \( F_{s} (\frac{m_{ij}}{m}) \) claims higher values if \( m_{ij} \) reflects tuples matching the same template that hold the majority of the tuple space (at least 50%). On the other hand \( F_{s} \) returns low values if tuples of the same kind in question are in the minority. The multiplication with \( m_{ij} \) is a weighting factor that expresses the importance of a tuple type. Thus it scales the concentration value \( C_{new} \) that influences the drop probability.

From the ants’ point of view, they tend to store a tuple in a tuple space if there is a clear proportion of tuples forming the majority. However, they are more likely to continue to carry the tuple and head towards the next tuple space if the current tuple space does not indicate high similarity.

IV. SIMULATOR

In order to investigate both visually and empirically the behavior of SWARMINDA we have implemented a simulator using the cross-platform multi-agent programmable modeling environment NetLogo [15]. The GUI for the entire SWARMINDA simulator is shown in Figure 2.

SWARMINDA works on the top of a scale-free network. Hence the first operation we have implemented was a generation of scale-free networks according to the preferential attachment rule in the so-called BA model [16]. The rule states that networks grow in a way that new nodes are connected to others according to the others current connectivity level of that node (degree)—the higher the degree the higher the probability that new nodes will attach to them. In our case we have used the preferential attachment rule that states that every new node attaches to at least two already existing nodes using the preferential attachment rule. The result of this that we have run our experiments in networks that are free of scale as stated in [16], Figure 1 shows an example of a scale free network generated using our simulator.

A. Modifiable Parameters

There are several parameters in the simulator that can be modified in order to achieve different behavior. Most of the parameters that follow can be seen in Figure 2.

Number of Agents: Sets the number of agents that will be instantiated. When the out or in buttons are pressed, the simulator instantiate a number of agents in the network to perform the respective operation.

Age: Sets the maximum age each ant has for fulfilling its task. This value is the \( ttl \) value determining the amount of hops an ant can take before it dies. The age value is assigned to an ant once it is instantiated. Modifying this value does not affect ants already in the network.

Type of Tuple: Sets the type of tuple the ant should carry when it gets instantiated. The type can be set to all so that the different types of tuples will be equally distributed among the number of ants. Note that an ant can carry only one tuple. Modifying the type of tuple affects only new instantiated ants.

Node Selection: Sets the location where ants should be instantiated. The type can be set to all meaning that ants will be created randomly across the network.

Clean-up Mode: Sets the mode for cleaning up the network. There is a specific type of ant that performs the tuple movement. The objective is to improve the level of order in the network. The tuple movement is composed of a Pickup phase and the Drop phase.

The Drop phase works similar to the tuple distribution mechanism explained earlier. The Pickup phase can be used in two modes: In the deterministic approach, clean-up ants are always picking up that tuple which has the lowest amount of hops around as long as the entropy level is greater than 0. Once the system is in a complete order the ants stop working. As a result of this process, minority objects should be moved to other regions where they fit better in the context of clustering. The other mode is the probabilistic mode where the ants works based on probability of picking up a tuple which in turn is based on the local entropy of a node and the conformity of the different tuple types on that node in contrast to the neighborhood. If the local entropy tends to be chaotic and a particular tuple type does not fit in the current node, the probability to pickup the tuple tends to be high. The result is that the odd tuple will be picked up by the end and dropped where it fits better (according to the tuple distribution process).

V. EXPERIMENTS

We ran several tests on the simulator applying the different formulas for calculating the drop probability and the entropy mentioned in section III. In the following we demonstrate the new system behavior by comparing the original and
modified equation and show the improvement in terms of tuple distribution and movement.

For the test runs we instantiated 1000 ants in total (250 ants carrying the same type since we support four different tuple types) but not at the same time since in a real system this is unlikely to happen. So we spread it over time with the following intervals:

- 40 ants (10 of each type) instantiated 5 times (200 ants total).
- 200 ants (50 of each type) instantiated 4 times (800 ants total).

Each ant above executes an *out* primitive. Each instantiation was executed in groups meaning that the next instantiation waits until all of the previous ants are done with their tasks. Therefore we have 9 iterations.

Figures 3, 4 and 5 show the average spatial entropy in relation of steps executed in the simulator. The curve exhibits the average of 20 test runs on the simulator. The whisker chart shows also the minimum and maximum and with it the range of the measured data points. The drawn gray areas in proximity to the curve shows the lower and upper quartile containing 50 percent of the data while the upper and lower whisker indicates the outliers.

In Figure 3 we applied Equations (1) and (3). The figure shows the amount of time needed for dropping all tuples carried by the ants in the network. Performing this process results in a light decrease of the entropy and the system does seem to be less sensitive to the organization of tuples in the network.

Compared to Figure 4 which uses Equations (2) and (6) the entropy increases to a very high level since the formula assigns nodes with more tuples with more importance. Empty tuple spaces are not contributing anything to the entropy since they are not used in the system as explained in Section III. The relatively larger range between the minimum and maximum value of the whisker in the beginning appears because the initial distribution of tuples in the first iteration is almost random. But focusing on the development of the graph one can see a steep decrease of the entropy, the quartiles and the range of the whisker. The entropy tends to go down very fast and finally converges.

However one has to discuss the seemingly bad initial results when applying the Equation (2). This takes place because it does not involve empty tuple spaces in the calculation as Equation (1) does. The argument here is not that our entropy calculation is worse than what was done before but rather that what was done before may not be realistic in all cases (as in the case with empty tuple spaces). Still, as shown in Figure 5, the curve representing Equation (2) intersects the one of Equation (1) at step 138 and continues to decrease. Thus we gain a better entropy value in the end because the distribution of tuples over the network is more balanced. With it we get an improved self-organization of tuples in terms of establishing homogeneous clusters on the nodes.

Figure 5 shows all four combinations of the original and modified drop probability and entropy. Comparing the lower two curves one can see that the one applying the modified drop probability has a steeper decrease of the entropy because of the more balanced distribution of tuples. In contrast, using the original drop probability results in a light decrease of the entropy. These two curves show only the differences between applying the original and modified drop probability since both of them are based on the original entropy computation. Similarly, comparing the upper two curves shows more significantly the different behaviors since they are based on the modified entropy computation. In more detail, one can observe that the most upper one indicates that it is getting worse around step 95 since different tuple types tend to stay at the same node.
resulting in heterogeneous clusters.

Figure 6 shows the results of tuple movement continuing after the distribution phase. The results were done starting from the situation right at the end of what is shown in Figure 5. One can see that it will take about 100 steps applying Equation (2) and (6) for pushing the entropy towards 0. In the remaining 800 steps there is almost nothing to do since the organization of tuples is already very good. This will decrease communication overhead and save bandwidth.

In a real system both out and in take place as well as tuple movement so the ability to reach good levels of organization after a short period of time can improve robustness and performance of the system. Note that this means that the application of Equations (2) and (3) are not interesting because it will take a long time for entropy of the system to reach acceptable levels. Even the 900 steps of the simulation are not sufficient enough: the entropy levels at the end of the simulation is 0.23.

Last, we considered the behavior of the system using Equations (1) and (6). As it can be seen in Figure 6 the entropy levels are not as low as desirable at the end of the execution. This is due to Equation (1) that treats nodes with a small amount of tuples in the same way they had large amounts (e.g. 1000). For instance if we assume we have at least one node with very few tuples but a bad entropy, this node contributes to the network entropy at the same level as nodes with large amount of tuples. Compared to the curves applying Equation (2) the contribution of the node entropy is weighted based on the amount of tuples stored at the tuple spaces (as described in section III).

It is very noticeable that in the beginning of the execution of the experiments in Figures 3, 4 and 5 the entropy value stays at 0 (zero), thus complete order. Around step 19 one can see a steep increase. This phenomenon is easy to explain. Since these tests were performed on an empty network, the tuple spaces are empty and there are no pheromone in the network. By applying either Equation (3) or (6) it is easy to note that if the concentration value \( C \) as well as \( C_{\text{new}} \) tends to be 0 the probability of dropping a tuple is almost 0, too. If the ants are roaming around in the network, the \( \text{ttl} \) value \( K \) is decreasing thus the drop probability increases a little but in most cases it is not enough for dropping the tuple. Finally if the ant reaches its maximum age, the \( K \) value reaches 0 (zero), the ant dies, and it stores the tuple in the current node. The change in behavior around step 19 relates to the fact that these experiments were run with the age value set to 20. At step 20, since the first ants were not influenced by pheromone, a more chaotic pattern can be observed resulting in a large entropy value. Soon after step 20, the ants can start using information present in the network such as tracking pheromone trails. This process leads to the formation of clusters and the decrease in the system entropy.
VI. Future Work

The obvious future work is the definition of the fourth algorithm as defined in SWARM-LINDA [7]. It has been originally argued that movement of tuples should be counter-balanced with possible movement of templates. In other words, the idea is that SWARM-LINDA should decide to move tuples of template depending of what is more advantageous. For instance, if many tuples of a particular kind are clustered in a location and a template is issued to retrieve a tuple, the template should perhaps move since it would be more cost-effective than moving all the tuples in search of this template. What this approach encompasses is that the decision of moving tuples or templates should be made as part of a self-organized approach.

Another idea to investigate is the issue of bulk movement of tuples. Some LINDA implementations assume the existence of primitives that are able to move or copy several tuples of the same kind at once ([17], [18]). One question that arises here is whether this bulk movement can be used as part of the agents responsible for tuple organization within SWARM-LINDA. Conversely, we need to see how these operations can be abstracted as ant-like agents roaming within SWARM-LINDA.

VII. Conclusion

The ability of coordination systems to work well on large scale environments is a condition necessary for these systems to become widely adopted in computer science. More importantly, in these large scale environments, centralization is a burden that should be avoided at all cost.

This paper has demonstrated the power of SWARM-LINDA, a distributed coordination model based on LINDA. The simulator enabled us to confirm that SWARM-LINDA demonstrates self-organized characteristics while allowing an easy visualization tool of the behavior of SWARM-LINDA.

It should be noted however that the main contribution of the paper refers to the study of the tuple movement behavior of SWARM-LINDA. Since its initial proposal, this algorithm has not been shown to yield good entropic characteristics. Here we have demonstrated that it indeed makes SWARM-LINDA be more self-organized. As a secondary contribution, this paper has refined the algorithms originally proposed as well as the way the system entropy should be calculated.

REFERENCES