



# A New Approach to Scalable Linda-systems Based on Swarms

**Ronaldo Menezes**

Florida Tech

Department of Computer Sciences

**Robert Tolksdorf**

Freie Universität Berlin

Institut für Informatik

# Introduction

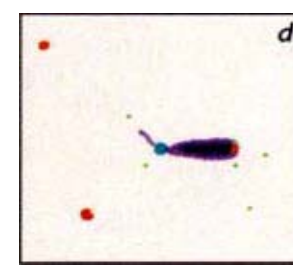
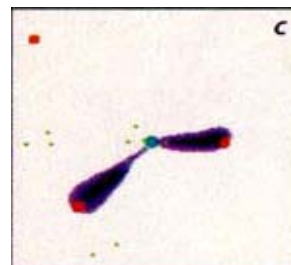
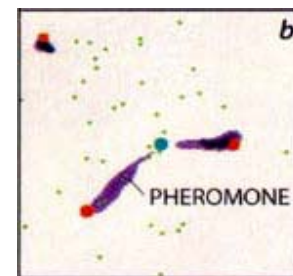
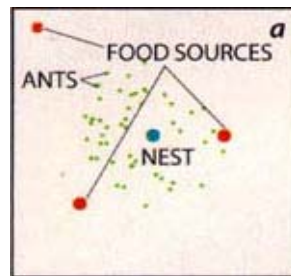
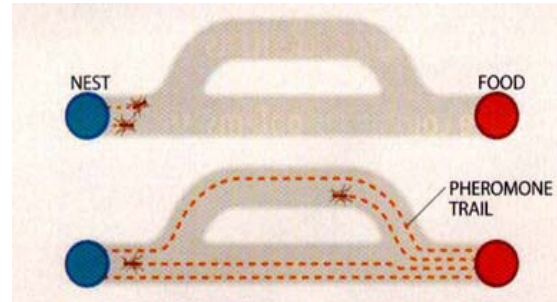
- There are a plethora of Linda-based systems out there
- Why another one?
  - Scalability is a central problem in Linda systems
  - Adaptability is almost non-existent
  - Locality is not well explored
- Where can we find solutions to the above?
  - Natural-forming multi-agent systems (aka Swarms)

# Self-Organizing Systems

- Nature has demonstrated to us how large-scale systems may be organized
  - Examples range from swarm of bees to colony of ants
- Despite their differences, they share several interesting characteristics
  - Scalability
  - Adaptability
  - Emergence
  - Decentralization
  - Convergence

# Example Adaptability of Swarms

(Bonabeau & Théraulaz, 2000)



# Swarms

- Swarms provide an abstraction that can be used on the development of distributed, large-scale, systems
  - Individuals are simple
  - Extreme decentralization: There is no global state, decisions are based on local information
    - No single entity is “in charge”
  - Collective behavior emerges from simple interaction among the individuals
  - The collective behavior adapts to changes in the environment
  - Convergence to (near)optimum is guaranteed

# Scalability in Linda Systems

- Theoretical works
  - Demonstrate how better scalability can be achieved from the organization of tuples [Obreiter & Gräf]
  - This is generally implemented through the mechanism of hashing of tuples
- Practical Works
  - Concentrate on the organization of tuples spaces [Rowstron]
- However
  - Good scalability is not yet a fact in Linda systems.
  - Lack of applications in the realm of *large scale* Linda systems only makes it more difficult to study scalability.

# Typical Scenario

- Problem
  - Retrieve a tuple matching a given template from a set of remote servers
- Possible Solution
  - Request the tuple from a (sub)set of servers via some multicast
  - Each server searches, matches and locks the tuple while it offers it to the requester
    - The tuple is an “under request” kind of (global!) status
- Does this work if the system is *very* large!

# Distributing Linda

- Different approaches to tuple space distribution in Linda
  - Centralized implementations
  - Partitioned implementations
  - Use of full replication
  - Partial replication



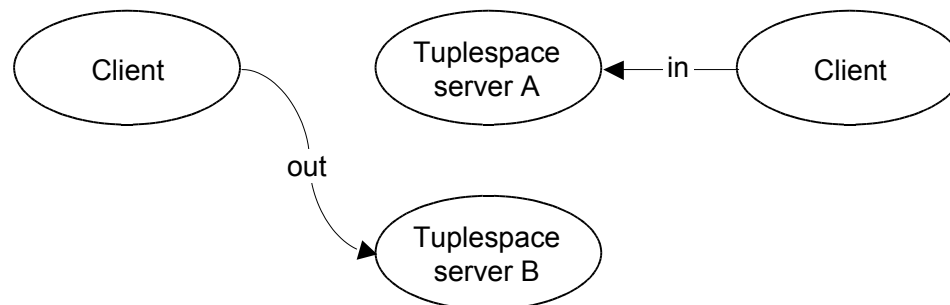
# Centralized Tuple Spaces

- This has also been called distributed central servers implementation
- Servers contains a tuple space in its entirety
- Advantages
  - Easy implementation
  - Simpler to include fault-tolerant features
- Disadvantages
  - Tuple spaces (or the servers they are stored) may become bottlenecks



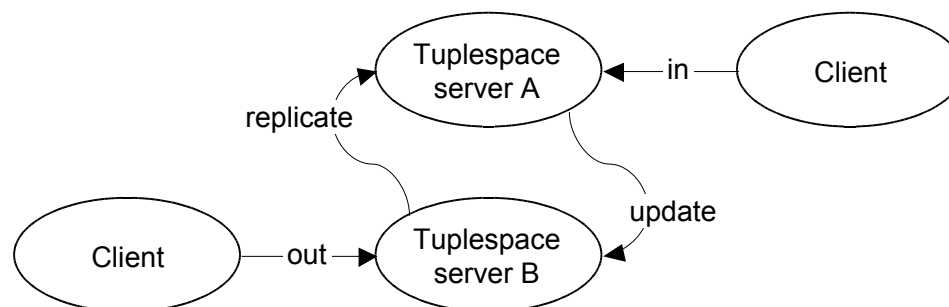
# Partitioned Tuple Spaces

- Here, tuples are distributed across servers based on their inherent characteristics (ie. parity, type of fields)
- Advantages
  - Deals better with concurrent accesses to tuple spaces
- Disadvantages
  - Bottlenecks are still possible
    - Good hashing techniques might improve the situation
  - Difficult reconfiguration. That is, poor adaptability



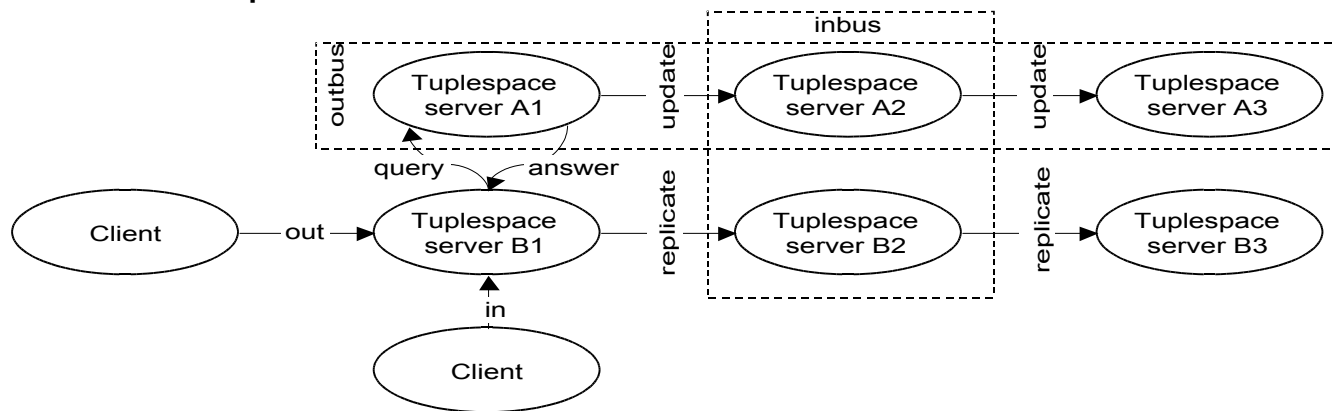
# Full Replicated Tuple Spaces

- Here entire tuple spaces are replicated across different machines
- Advantages
  - Load sharing in the context of tuple searches
- Disadvantages
  - High communication overhead
  - Poor scalability



# Intermediate Replicated Tuple Spaces

- Uses the concept of virtual in-buses and out-buses
  - Requests are performed in the in-buses while store of tuples are replicated in the through the out-buses
- Advantages
  - Allows for more concurrency
- Disadvantages
  - Overhead of maintaining replicas
  - Difficult implementation



# Concepts of a SwarmLinda

- **Simplicity**
  - The implementation should be simple
  - Organized behavior should emerge from simple interactions between the entities in the system
  - Activities should be small in terms of resource usage
- **Dynamism**
  - The environment configuration may change and entities should react to the changes
    - Convergence to the new configuration should be done without adding complexity to the system
- **Locality**
  - Decisions must be local to minimize communication overhead

# Algorithms for SwarmLinda

## (Distributing Tuples)

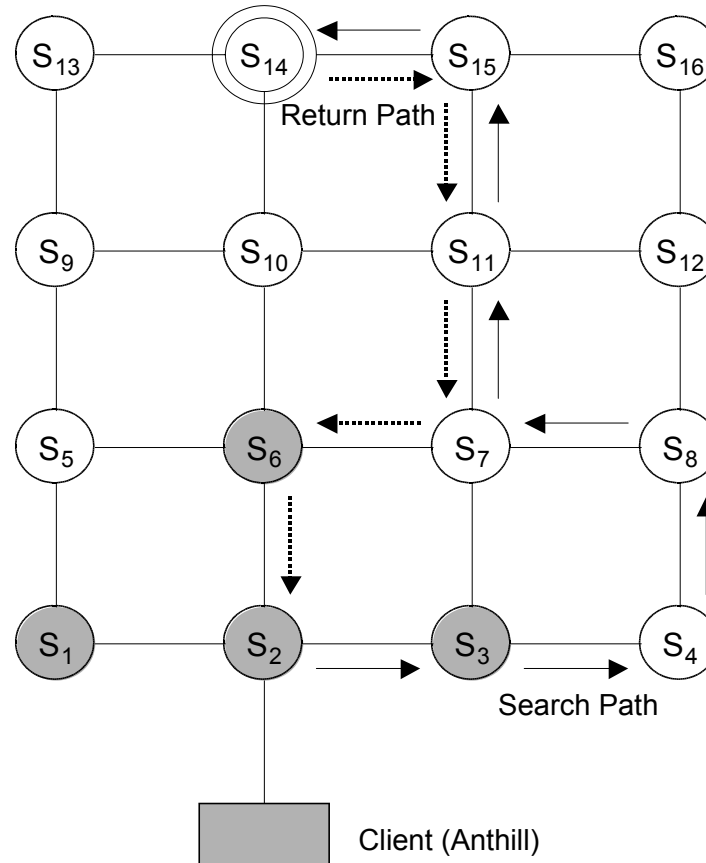
1. Start visiting servers upon execution of an out primitive
2. Observe the “kind” of tuples the servers are storing
  - No need to remember the status of the entire system
3. Store tuple if nearby servers are storing similar tuples
  - Decision is based on a random factor  $[-\xi, \xi]$
4. Move to next server if nearby servers do not contain similar tuples
  - Again, influenced by  $[-\xi, \xi]$

# Algorithms for SwarmLinda

## (Searching for Tuples)

1. Scent of the requestor is spread
2. Check for tuple in current location return to origin dropping a scent typeset by the tuple format. If a tuple is not found check servers in the neighborhood
3. If scents cannot be perceived nearby make a random choice. Otherwise use the strength of the scent as probability factor to move in that direction
  - New paths can be discovered
4. Activity of an ant (template) has to be limited so that it doesn't search for tuples not yet produced
  - The template has a probability  $\gamma$  of stopping the search
5. Activity can resume at a later time

# Searching for Tuples





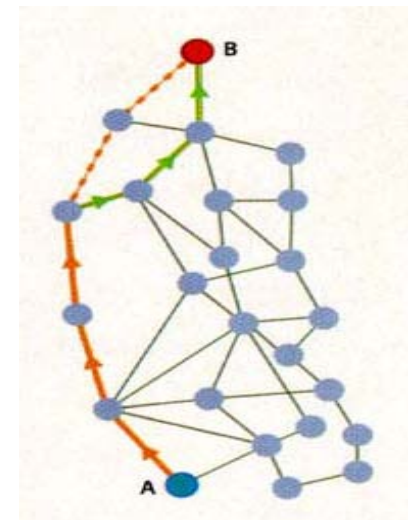
# Algorithms for SwarmLinda

## (Dealing with Openness)

- Given a function  $Sc: T \rightarrow S$  and a relation  $C: S \times S$ .  
If a template  $te$  matches a tuple  $tu$ , then  
 $Sc(te), Sc(tu) \in C$ 
  1. New tuples emit  $Sc(tu)$  and new templates emit  $Sc(te)$
  2. Tuple-ants are always sensing their environment trying to find similar scents – as defined in  $C$
  3. Based on the strength of scent on the nearby servers the tuple-ant decide to move to that location or to stay where it is
    - This causes tuples to stay closer to where other similar tuples are needed or are being produced.

# Qualitative Discussion

- **Scalability**
  - Decisions are local
  - No global state (eg. locking a set of servers)
- **Adaptability**
  - Decisions are based on the current state of the system which in turn can change
- **Fault-Tolerance**
  - Availability of the system is improved
  - Effect of wrong decisions is temporary
- **Load Balancing**
  - Bottlenecks are avoided



(Bonabeau & Théraulaz, 2000)

# Implementation

- Tuple spaces don't exist as structures
  - From the point of view of the process this would not be apparent as the primitives are still typeset by the tuple space name.
    - A handle is created but not the space itself
  - This is a consequence of the tuple distribution algorithm of SwarmLinda that does not “care” about the location of the tuples
  - Tuples may exist anywhere in the grid of servers
- Servers organized in two-dimensional grids
  - Allows for easy reconfiguration of server neighbors and merging of separate grids
  - Servers are active in the sense that they are responsible for maintaining scents
- Other topologies may be used

# Conclusion

- SwarmLinda uses out of the box solutions
  - No global state
  - Easy implementation
  - Dynamic reconfiguration
- SwarmLinda Website
  - ([cs.fit.edu/~rmenezes/SwarmLinda](http://cs.fit.edu/~rmenezes/SwarmLinda))
  - ([www.inf.fu-berlin.de/inst/ag-nbi/research/swarmlinda](http://www.inf.fu-berlin.de/inst/ag-nbi/research/swarmlinda))
  - Papers
  - Presentations
  - Prototypes

# Questions?