TECHNISCHE UNIVERSITÄT WIEN
Vienna University of Technology

Diese Dissertation haben begutachtet:

---------------------   ----------------------

DISSERTATION

SELF-ORGANIZATION FOR LOAD BALANCING AND INFORMATION RETRIEVAL BASED ON SHARED COORDINATION SPACES

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Doktors
der technischen Wissenschaften unter der Leitung von

Ao. Univ.Prof. Dr. eva Kühn
185-1
Institut für Computersprachen

und
Univ.Prof. Dr. Slobodanka Mitrovic
Universität Belgrad

eingereicht an der Technischen Universität Wien
Fakultät für Informatik
von

Mag. Dipl.Math. Vesna Čavić geb. Šešum
0625918
Mohsgasse 26/15-17
A-1030 Wien

Wien, 10.02.2011.   eigenhändige Unterschrift

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43/(0)1/58801-0 http://www.tuwien.ac.at

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF LISTINGS

ABSTRACT

The increased complexity in nowadays information technology (especially in distributed systems) presents a huge obstacle in the further development of software systems. The huge number of unpredictable dependencies on interacting components cannot be coped with any more in a traditional way. It implies the necessity of finding more advanced, intelligent approaches. As software systems develop rapidly and change constantly, the existing methods are obsolete or inadequate in today's dynamic environments. Therefore, self-organization that is a relatively new approach with a lack of real applications is proposed in the dissertation as a promising approach in coping with complexity. The dissertation presents a new conception of a self-organizing coordination infrastructure as a combination of different methods: coordination spaces, self-organization, adaptive algorithms and multi-agent technologies. The focus is put on two important IT problems: dynamic load balancing in heterogeneous distributed systems and information retrieval in the Internet. These problems are treated in a new way by using self-mechanisms. For each of these problems, a self-organizing framework, i.e., software architecture is developed. These architectures are modeled and their correctness is proven by using the PlusCal algorithm language. They are flexible and generic, undependable of the network topology, algorithms used, problem specification, etc. A bee algorithm and two adapted ant algorithms are developed for the located IT scenarios. These algorithms are inspired by self-organization from nature. They mathematically describe bio-self-mechanisms and successfully solve these complex problems through autonomy and fully distributed communication of components in a system. These algorithms are plugged in the frameworks. The results are obtained by benchmarking in two different environments: a cluster and the Amazon EC2 Cloud. The benchmarking part presents a way of selecting and fine-tuning of a huge number of parameters used in the algorithms. The comparison is done taking into account the other approaches: Gnutella lookup mechanisms for the information retrieval in the Internet, and different unintelligent (Random, Sender) and intelligent (adapted genetic algorithms) approaches for dynamic load balancing. The evaluation is carried out by performance and scalability. The obtained results prove the benefits of the used methods and constructed algorithms as the performance of the system and scalability are improved. For example, the results of the first considered scenario obtained on the Amazon EC2 Cloud, showed that the random/bee combination on 80 nodes with 50 swarms and by treating 5 queries was 0.5% better than the random/AntNet combination, 7.8% better than the random/MMAS combination and 61.3% better than Gnutella. The results of the first considered scenario obtained on the Amazon EC2 Cloud, showed that: in the *chain* topology, the best result is obtained by both BeeAlgorithm/Sender and MMAS/MMAS. They were equal good, and better than the combination that “took the second place”, GA/Bee Algorithm, for 5.4%. The combination RoundRobin/BeeAlgorithm showed the best results in the *full* topology. This combination was better than the combination that “took the second place”, RoundRobin/AntNet, for 1.3%. Both BeeAlgorithm/Sender and MMAS/RoundRobin were equal good in the *ring* topology. They were better than the combination that “took the second place”, MMAS/RoundRobin, for 1.4%. In the *star* topology, the combinations BeeAlgorithm/BeeAlgorithm and GA/AntNet were the best with the same resulting value. They were better than the combination that “took the second place”, AntNet/MMAS, for 6.1%. The self-organization is measured through the usage of specially constructed functions (so-called the suitability function). The main innovation and contribution of this dissertation is: location of problem types where self-* can be useful, construction of a new self-organizing coordination infrastrucutre, adaptation of Ant Algorithms for the located IT problems, specification of a new type of algorithm, Bee Algorithms for the located IT problems, finding the best parameters tuning in each of the considered scenarios as well as the best algorithm/combination of algorithms.

CHAPTER 1

1 INTRODUCTION

Self-organization surrounds us and offers apparently simple answers to very complex questions and problems through spontaneously increased organization in a system without actions being controlled by some central coordinator or an external system. It could be seen as a natural process of evolution through which complex systems find qualitatively better patterns in order to cope with their complexity. These mechanisms are extremely interesting to be applied in different complex systems nowadays, especially IT-systems. This chapter starts with a major issue in IT industry, the increased complexity of software systems. Then it points out the advantages of self-organizing mechanisms and necessity of their usage in IT problems. After having introduced the research area, this chapter continues with deriving the main research questions, explanation of the chosen approaches and the ways of evaluation of the achieved results.

1.1 Complexity in IT systems

Today's information technology industry is characterized by the growing globalization of enterprises. In order to be competitive, companies require software systems to communicate and collaborate across organizational boundaries, using software components providing data and services from many different, distributed, and heterogeneous sites. Such software systems are rapidly changing caused by new and varying market needs, as well as by technological evolutions. Developers of distributed software systems put a significant effort to keep their systems up-to-date with all new standards and, therefore, have to cope with an enormous increase in software complexity. Main factors that determine software complexity are: huge amounts of distributed components that must interplay in a global solution, problem size like number of computers, clients, requests, size of queries etc., heterogeneity, autonomy of organizations, and dynamic changes of the environment. Distributed software systems are forced to integrate other software systems and components that themselves are often not reliable, exhibit bad performance, and are sometimes unavailable. As they are run and maintained by autonomous organizations they neither can be changed, nor adapted, nor hosted elsewhere.

These challenges are so fundamental that the usually taken approach to control distributed components across enterprise boundaries through one central coordinator software reaches its technical and conceptual limits. It is hard to design such a single controlling component that is à priori aware of all the mentioned possible changes and deficiencies in the environment. The more foreign software components are involved, the more the risks concerning complexity of utilization, entire system performance, and operational dependability, increase. Components and their capacity increase exponentially and overall-complexity increases super-exponentially ([HeGe03]). Therefore the huge number of unpredictable dependencies on participating components cannot be coped with any more in the traditional way, namely through one central coordinator that implements the entire business logic and that possesses the complete picture of the distributed environment. Rather, completely new approaches are demanded to diminish these problems.

Generally, complex systems are systems featuring a large number of interacting components with internal state defined by a huge number of parameters. These systems can be in any of a very large number of states at any given time. Complex systems behave unreliably, with a number of unexpected and often unexplained upturns. Except unpredictability, the behaviour of a complex system is characterized by non-linearity, asymmetry and aperiodicity.

It is difficult to "decompose" and analyse such systems with a huge number of elements (often heterogeneous) and relations between those elements, whereas the way of decision–making is highly decentralized. If complex systems are so unpredictable, how can we deal with them? We are not able to deal completely with them and can not predict everything that might happen. What we can do is to be prepared to adapt as good as possible to the unexpected changes, and to anticipate as much as we can. When a system is adaptive, unexpected events can be tackled, as the system is reconfigured or reconfigures itself without breaking. The complex system is not arbitrarily regulated. It is ordered in a very organized way. This organization was not built into the system at its origin, it has emerged in a sequence of self-organizing processes that understood spontaneous transitions into new states of higher organizational complexity. The term "spontaneous" doesn't mean "they just happen" for no particular reason. The challenge and need is to find some principles additional to the low level laws to explain it. Some well-known methods used to deal with complexity (like simplicity, abstraction, decoupling, decomposition, classification, etc.) have been proven to be appropriate to handle specific problems. A very useful concept in the adaptation of complex systems is self-organization. Certainly, self-organizing systems will not be able to adapt to all possible events, but they have proven to pose a good perspective to deal with complexity. The goal for the elements of a system is to self-organize, without the intervention of an engineer or manager. The advantage of self-organizing systems is not only that they can find un-foreseen solutions for problems, but also that they are very adaptive. Major advantages over traditional systems are: robustness, flexibility, capability to function autonomously while demanding a minimum of supervision, and the spontaneous development of complex adaptations without need for detailed planning. Disadvantages are limited predictability and difficulty of control. These systems are “on the edge” between organization and chaos, which bears a certain risk.

The origin of the term “self-organization” dates from the 18th century and the work of Immanuel Kant ([Kant1892]). In contemporary science, the term "self-organizing" was introduced in 1947 by W. Ross Ashby ([Ashb47]) and later used by Norbert Wiener ([Wien61]). The notion of self-organization was used in the area of general systems theory in the 1960s, but the common usage in the scientific literature started its adoption by physicists and researchers in the field of complex systems in the 1970s and 1980s.

There is a constant necessity for self-organizing mechanisms in distributed systems. Researchers have experimented with different paradigms in order to achieve the main properties of self-organized systems. The paper ([MMTZ06]) presents a review on the state of the art of nature-inspired self-organizing mechanisms in computer science. Five main areas are currently identified to benefit from the presented self organized behaviours: middle-ware, information systems and management, security, robotics, and network management. Self-organization in *middleware* is further divided in four application areas: grid computing, coordination systems, cache replacement systems, and pervasive computing. In grid computing, the resource utilization needs to be adaptive to cope with dynamic conditions. Therefore, self-organized mechanisms such as foraging, molding, and brood sorting could be beneficial for the resource utilization in Grid frameworks ([AGKT02]). In coordination systems, the problem of scalability is present, i.e., it does not scale well with the number of processes in the system. SwarmLinda ([MeTo03]) tries to cope with this problem by applying the foraging and brood sorting mechanisms. In adaptive web cache replacement, newly created ants could follow such pheromones to predict what resources will be accessed in the future, and to crate novel relevant entries in the cache ([Floy05]). In pervasive computing, the models ([MaZL04], [MaZa04]) use the self-* mechanisms. Self–organization in *information systems and management* generally refers to database organization. In database organization, self-organization inspired by brood sorting that may be used to provide an adaptive distribution of data based on criteria based on the database tables, records, and even fields. These concepts are very similar to the ideas of self-tuning databases presented in ([WMHZ02]). In the area of security, self-organization is used in malicious code protection and refers to using self-mechanisms similar to the immune systems ([Dasg99]), and distribution of security policies (e.g., [Mene05] describes a self-organized solution for policy distribution based on foraging with elements of molding). In *robotic systems*, the use of self-organization is a “hot” topic of intensive research. Several surveys exist covering this application of self-organization (e.g., motion coordination [PaBS04] and self assembly [SLTD02]). Applications of self-organization in *networks* refer to areas of mobile ad-hoc networks (ant routing algorithms that apply foraging [BoDT99] and gradient routing algorithm that takes inspiration form molding [Poor01]), sensor networks (directed diffusion [InGE00] is a routing algorithm that uses of mechamisms like molding and ant foraging), and amorphous computing ([AACH00]) inspired by morphogenesis metaphors. However, the problem is that we still do not have the appropriate language to speak and think clearly about self-organizing systems. Although the theory of self-organization has much potential, it has no enough practical applications yet. What are real use cases that can profit from self-*[1]? On which kind of problem can it be applied? Self-* is not applicable to all kinds of problems, e.g. for "not very complex" problems or for problems that have a deterministic best solution, other methodologies will be more suitable and less risky. There are many facets of self-* one can think of, but probably not all of them will have a mapping and contribution to a real software problem. Some examples are:

- Self-healing/repairing systems will provide primitives for continued execution when nodes or the network communication fails as well as support the "repair" of node configuration.
- Self-configuring systems will provide mechanisms so that the software will continue to work when nodes are added/removed during execution and that parts of the application can be upgraded "on-the-fly" from one version to another without interrupting execution.
- Self-tuning systems will provide support for coping with high and dynamically changing loads through load balancing.
- Self-classifying systems will enable different optimization; e.g. minimize search effort or reduce network costs by clustering nodes according to certain interests.
- Self-learning systems will provide means to adapt fast to changes by learning from history.
- We expect that self-* is applicable to those types of problems where we can clearly adapt known self-* mechanisms from nature, organizations, social domains, etc.

In order to investigate complex systems and benefits of self-* in reducing their complexity in an abstract, general way, the knowledge about particular scenarios must be collected. It is hard to predict what constitutes the "critical mass" of scenarios (a huge number of very significant scenarios) after which it could be possible to lift up our cognition about self-organization in a more abstract way. This dissertation investigates two important IT problems: load balancing in heterogeneous distributed systems and information retrieval in the Internet.

[1] Self-* denotes all possible self-properties of a system that can lead to self-organization.

1.2 Research Questions

Although the potential of self-organization in approaching to problems in complex systems is recognized, there are many open questions in this field of research. This subsection describes open problems and derives the research questions addressed in this work.

Research Question 1: *Can the two important IT use cases: 1) load balancing in heterogeneous distributed systems, and 2) information retrieval in the Internet, profit from the usage of self-organization?*

Although self-* has much potential and current research on it has raised reasonable interest, there is a lack of real-life applications relevant enough to derive a substantial practical experience from them ([Heyl01]). Self-* has been successfully applied to combinatorial optimization problems and to problems that treat routing, or search and optimization in general ([DoSt05], [Blem03], [Stüt97], [DiDo98a], [DiDo98b], [ChZC10]) or the clustering or grouping/aggregation of data ([CMVT07], [TaVe05]). But definitely, self-* is not applicable to all kinds of problems, e.g. for "not very complex" problems or for problems that have a deterministic best solution, other methodologies will be more suitable. There are many facets of self-* one can think of, but probably not all of them will have a mapping and contribution to a real software problem. We expect that self-* is applicable to those types of problems where we can adapt already known self-* mechanisms from nature, organizations, social domains, etc. A mission of Research Question1 is to investigate two important IT problems: load balancing in heterogeneous distributed systems and information retrieval in the Internet and find out, whether and to which degree they can profit from principles of self-*.

Research Question 2: *Can the principles of self-* help to cope with complexity in heterogeneous systems? What can be improved by employing self-* mechanism? What kind of complexity exists and how can complexity be measured with the focus on the above mentioned problems?*

Each software system bears a certain degree of unavoidable complexity which refers to its specification ([Alha04], [KuKG08]). The kind of software systems to be investigated in this research work are characterized by a huge number of heterogeneous, distributed, unreliable components that need to collaborate to achieve a joint goal, dynamic changes in the environment, and complicated business requirements (e.g. complex queries). There is often not one "best" solution, rather there are many possible good or at least acceptable solutions. A repeated execution of a software application might lead to different results due to changed environmental conditions. Research Question 2 investigates whether decentralized, autonomous adaptation of component behaviour, which allows a fast reaction to any kind of changes and opportunities, is a proper means to cope with complexity in heterogeneous software systems. This refers on the one side to the issue to become able to build solutions that are thought infeasible today as their design and development effort would be too high, and on the other side the issue of reducing operational risks as a self-* system is able to heal itself automatically (e.g. through failover mechanisms). An example of a self-* system is the Internet where there is no central control, each node of the network has its own task, and the Internet protocol is designed in a way that if some servers go down, the traffic can be still maintained by other servers. The Internet adapts constantly to varying traffic loads. The rationale behind this research question is that modelling and deployment of smaller and self-contained software components is easier than designing a complex system in its entirety. Central responsibility, knowledge about all possible circumstances from the outset, and a single-point-of-failure are avoided. Unexpected events are tackled, as the system reconfigures itself without breaking. A running system evolves to a superior solution by itself without explicit intervention of a software developer. The advantage is both to find unforeseen, good solutions for complex problems, and to adapt and improve automatically whenever possible. Certainly, self-* systems will not be able to adapt to all possible events, but we believe they promise a good perspective to deal with complexity. It must be investigated, whether new sources of complexity arise, how to guarantee that a system finds a solution at all, will the system spend too much time "administrating" itself, and can certain service level agreements be met.

Researchers from different areas of science like biology, computer science, finance, etc., define different measures of complexity for each respective field. ([Lloy01]) presents a categorization of complexity measures by defining common questions for all problems: (1) How hard is to describe? (2) How hard is it to create? (3) What is its degree of organization? Obviously, a general form doesn't exist yet; e.g., in ([CMVT07]), the mechanism of "brood sorting" is used and as one measure a kind of spatial entropy is proposed. In ([ŠeKü08], [ŠeKü09]), it is tracked how good the single contributors (bees, ants, …) organize themselves by means of suitability functions.

Also, it will be investigated how the application of self-* could improve performance and scalability of a system both scenarios.

Research Question 3: *How can swarm intelligence be mapped/adapted to the load balancing problem and to the problem of locating and retrieving information in the Internet? Can bee intelligence be mapped to these two use cases and how? Can ant intelligence be adapted to these two use cases and how?*

As there is not yet a broad practical experience in the field of distributed self-* software systems, Research Question 3 claims a need for (a) identifying existing tools, methods, and architectures capable to be applied in the self-* field, and (b) investigating and developing new ones. An appropriate language to speak and think about self-* systems is required, as well as methods to evaluate a self-* system at both theoretical and practical level. Specific attention will be given to swarm intelligence mechanisms (ant intelligence, bee intelligence) as a very promising approach to obtain self-* properties ([DoSt05], [WoLC08]).

Research Question 4: *What is the best parameters tuning in each of the considered scenarios?*

Generally, dynamical systems are very sensitive to parameter changes. For example, a single mutation leads the system into another completely different behavior. As the possible states grow rapidly with complexity, dynamical systems possess very large state spaces. During these changes of state, a system moves to a fixed structure, i.e., it arrives at the attractor - a preferred position for the system ([Heyl01]). When we are talking about "transferring" self-* mechanisms from nature (like the usage of swarm intelligence), the proper parameter settings and fine-tuning is a very delicate task.

Research Question 5: *Is it better to have an intelligent approach or an unintelligent approach or a certain combination (which one)?*

Intelligent approaches are new, promising ones. The investigation includes: (a) whether it is always true or not true that intelligent approaches (or a certain combination or hybrid) could outperform unintelligent ones, and (b) what are specific situations in which intelligent approaches "win", i.e., what the intelligent approaches' success depends on (e.g., a certain network topology, etc).

1.3 Approach and Contribution

This thesis presents a new conception of a self-organizing coordination infrastructure that suggests a combination of coordination spaces, self-organization, adaptive algorithms, and multi-agent technologies. Each of the numbered issues has some form of self-organization in their incentives. For the approach of this work, a finite set of self-* properties are considered that are useful for the establishment of self-organizing coordinating infrastructures. The intention is to develop a guideline for classification of self-organizing systems according to the specified set of self-* properties. It should be mentioned that it is not the goal to limit the number of self-* properties, i.e., systems should have as much features based on self-* functions as applicable and needed. Distributed complex IT systems, i.e., coordination model(s) that contribute successfully in parallel systems' applications are connected with complex adaptive systems by mapping underlying mechanisms.

Chapter 3 (as well as Chapter 6) addresses research questions 1 and 2. The location of problem types where self-* can be useful is the starting important step. Using a thorough interdisciplinary literature search of use cases in different domains, the characteristic scenarios are located. NP hard problems (or problems that include some type of combinatorial optimization problem), where searching and optimization is necessary to perform, can benefit of self-*. Also, clustering of data can benefit of self-*. As a contribution, two well-known distributed systems' scenarios: load-balancing in heterogeneous distributed systems, and searching, retrieving as well as placing information in the Internet, are investigated. Further, the problem is approached by employing principles of self-organization at different levels in software architectures, and shifting the complexity from one central coordinator component to many distributed, autonomously acting software components. These components optimize their behaviours in a dynamic, ad-hoc way and thus adapt quickly and self-subsistent to both changing requirements and dynamically evolving system states. The latter are caused through the interplay and contribution of the many components to a global goal. Emphasis is put on the performance and scalability of the solution, and the work is positioned in the scope of heterogeneous distributed peer-to-peer systems. The contribution is location of type of complexity in the considered application case and explanation of possible complexity measurements.

Chapter 4 prepares basis and frameworks for giving answers to research questions 4 and 5, and partially answers them. Namely, two self-organizing coordination architectures on the pattern layer are developed with that purpose: SILBA (which stands for self initiative load balancing agents) for load-balancing in heterogeneous distributed systems, and another, simpler architecture for searching, retrieving and placing information in the Internet. SILBA framework comprises a realization of the complex and advanced mechanism for load balancing problem extended on several levels, i.e., load balancing problem is solved on an abstract way that can be transferred on the higher level. The novelty for the case of information retrieval scenario is a definition and an implementation of a new overlay network with an intelligent lookup mechanism based on swarm intelligence that is able to navigate successfully through the network of data and scales well. The models are described in the PlusCal algorithms languages and their correctness is proved via TLC model-checker ([Lamp09]). A detailed and comprehensive fine tuning of parameters is applied in both applications' scenarios as well as construction of combinations of algorithms and their hybrid forms.

Chapter 5 addresses research question 3 and explains how swarm intelligent can be mapped or adapted to the located application cases. In a certain sense, we cannot "invent" new forms of self-*. It already exists around us. We must learn from it – biologically-based mechanisms and emergences of forms are good examples – and try to transfer and implement such mechanisms into software systems. Such systems have the following advantages over traditional systems: robustness, flexibility, capability to function autonomously while demanding a minimum of supervision, and spontaneous development of complex adaptations without need for detailed planning. The contribution comprises a construction of a bee algorithm for load balancing and information retrieval, and an adaptation of two ant algorithms for load balancing and information retrieval. The novelty is the implementation of bee intelligence for the load balancing problem for the first time in order to improve the quality of the solution and scalability.

Finally, Chapter 6 answers research questions 4 and 5 as well as one part of research questions 1 and 2 (how and in which extent the employed principles of self-organization improve performance and scalability in the two application scenarios). For load balancing application scenario, it is explained why and where bee intelligence outperformed other (un)intelligent approaches taking in consideration the quality of a solution, the metric used and a scalability issue. It is detected which combination of algorithms fits the best to a particular network topology; also, detection which topologies profit the most from the application of swarm intelligence (by means of the used metric and scalability) is investigated.

1.4 Methods

The methods to be applied for a new conception of self-* coordination infrastructures comprise a combination of: shared data spaces, intelligent and adaptive algorithms, multi-agent technologies and benchmarking.

- *Use space-based computing as agile software architecture*

The distributed shared memory paradigm (also referred to "space-based computing") serves well for coordination of parallel and distributed processes ([PaAr98]). The main representative is the tuple space model ([CaGe89]). In our approach, we use a space-based architecture, called extensible virtual shared memory (XVSM) that generalizes Linda tuple based communication by more powerful coordination capabilities ([KüMS08]) and by extensibility of behaviour through aspects ([KMKS09]). Space-based middleware uses a blackboard based communication for the interaction of autonomous peers. Spaces have proven to be useful for communication between autonomous agents ([KMKS09]). An agent can organize its behaviour by accessing the shared state where it finds information about the environment, so it can decide by itself what information to pull or to be notified about and what to do afterwards. A state is needed to remember history. The idea is to shift complexity from a global "instructor" to smaller and autonomous pieces, i.e., the behaviours of single agents.

- *Adopt and develop intelligent algorithms; apply different algorithm in combination and/or form hybrid algorithms*

The intension is to learn from nature, prolific with self-* mechanisms, to detect, map and adapt these mechanisms, and apply them to real computer science problems. In mapping, software agents will play the role of a particular swarm (e.g., ants, bees) and "perform" self-* actions characteristic for the respective bio-colony. All these mechanisms are characterized by a huge number of different environmental parameters influencing the behaviour of artificial swarms (e.g., [DoSt05]). Therefore, it is very important to find out best possible combinations of parameter settings of the algorithms for the given use cases (i.e., there must be a context to define the best parameters).

- *Use autonomous agents and multi-agents technologies*

In agent-based systems ([ShLe09]) an agent is an entity (e.g. software module) that acts or has power or authority to act and cause changes. Its autonomy implies that its actions are neither controlled by others nor by outside forces. It is independent in mind, judgment or government, it is self-directed and self-governing. An agent acquires sensory data from its environment and decides by itself how to relate the external stimulus to its behaviours in order to attain certain goals. Responding to different stimuli received from its environment, the agent selects and exhibits different behavioural patterns. These may be predefined, or dynamically acquired by the agent based on learning and adaptation mechanisms. In a single-agent setting, it must understand high level goals and have knowledge about its abilities. In a multi-agent setting, it must have some idea about the other agents and ways to communicate and collaborate with them to share knowledge. The power of autonomous agents lies in their ability to deal with unpredictable, dynamically changing, heterogeneous environments. Therefore, intelligent algorithms benefit from autonomous agents. Autonomous agents that operate in a peer-to-peer network shall take over different roles (ants, bees, etc.) in the proposed research work. They are self-responsible to be up and running, implement a certain reactive and continuous behavior, and can dynamically join and leave.

- *Implement prototypes as proof-of-concept and perform benchmarks in real environments*

The space-based technology will be used for implementing prototypes. The creation of test examples will include a special attention to the fine tuning of parameters (as the intelligent and adaptive algorithms have many different configurable parameters that are mainly problem-sensitive). For the realization of benchmarks, two different test environments are available in order to investigate the behaviour of systems and algorithms. First test environment is a cluster of 4 machines at the Institute of Computer Languages at TU Vienna. Each machine has the following characteristics: 2*Quad AMD 2,0GHz with 16 GB RAM. Second test environment is the Amazon Cloud ([ACloud11]).

The following steps are taken to evaluate the concepts and methods proposed in this thesis.

As a first proof of concept, two prototypical implementations are developed: one for load balancing and another one for information retrieval.

A new framework termed SILBA is proposed and developed in this thesis as a generic architectural pattern for a load balancing that allows for the plugging of different load balancing algorithms, (reaching from unintelligent to intelligent ones) and foresees exchangeable policies for load-balancing. The presented pattern can be composed towards arbitrary network topologies and assumes autonomous agents and decentralized control. Further, SILBA is extended on several layers that allow routing between different subnets, simultaneously with load balancing between nodes within these subnets. Each network level can apply different algorithms and load balancing in the whole network will be realized through the combination of algorithms. Benchmarking is realized by using both environments.

For the case of information retrieval scenario, a new overlay network with an intelligent lookup mechanism is developed and implemented in this thesis. The chosen overlay is a purely decentralized and unstructured one (for an initial construction, the scale-free network approach is used). It supports a self-organized approach that combines a purely decentralized unstructured P2P system with space based computing in order to locate effectively and filter (retrieve) information from a network. The lookup mechanism is inspired by swarm intelligence (both ants and bees), is fully distributive and autonomous. Benchmarking is also realized by using by using both environments (a cluster and the Amazon Cloud).

1.5 Thesis Structure

This thesis is structured as follows:

Chapter 2. Technical Background and Related Work provides an overview of the state-of-the-art of technical concepts employed in this work, and explains the basis concepts and the theory on which they are established.

Chapter 3. Application Scenarios describes scenarios, located to be suitable for the appliance of self-* mechanisms, analyzes types of complexity that exist in these scenarios and ways of its measurement.

Chapter 4. Design and Implementation describes the frameworks used, the ways of their construction and implementation, and provides the proofs for correctness of the constructed architectures.

Chapter 5. Employing Nature-Based Mechanisms gives a detail explanation of the used swarm intelligent algorithms that are adapted and mapped to the application scenarios. A theoretical establishment is also discussed.

Chapter 6. Benchmarks explains the ways of generating test examples, fine-tuning crucial parameters, combining different algorithms on different network topologies, and evaluates the obtained results that are compared at the end.

Chapter 7. Conclusion summarizes approaches, contributions and results of this thesis, and describes future research directions.

CHAPTER 2

2 RELATED WORK AND TECHNICAL BACKGROUND

Related work focuses mainly on self-organization. It comprises the theoretical basement of self-organization in general, state-of-the art of the applications of self-mechanisms from nature and society, and the application of self-organization in P2P systems and in space based computing technology. The main technical background connected with this work refers to distributed systems, i.e. more specifically peer-to-peer (P2P) systems, and coordination models. At the end of this chapter, one section is dedicated to a short review of special types of algorithms – metaheuristics.

2.1 Self-Organization

Although we are surrounded with self-organizing mechanisms, the interest for an exploitation of them as well as the scientific study of self-organizing systems is relatively new, grown out of many scientific fields. However, a core of fundamental concepts and principles that should be applicable to all self-organizing systems has slowly started to emerge. The most popular self-mechanisms are those ones emerged in nature, detected and described by "exact" scientific disciplines (biology, physics, chemistry, mathematics). From the other side, the self-mechanisms exist also in social sciences (e.g., economics, collective intelligence, even linguistic). The scientific study of self-organized systems tries to discover the general rules of appearing self-organization as well as the forms which it can take.

This subsection starts with the description of some theoretical basement of self-organization in general and continues with the application of self-organized approaches in IT, originated both from nature and society.

2.1.1 Theoretical Overview

There are many definitions of self-organization. Some of them are provided below. After introducing a definition of a self-organization, a description of self-organizing mechanisms and an explanation of self-organizing characteristics are presented.

Def1([CDFSTB01]):

"Self-organisation of a system means that system structure appears without explicit pressure from outside the system and results from the interactions between the components, whilst being independent of the physical nature of those components. In general, it refers to the various mechanisms by which pattern, structure and order emerge spontaneously in complex systems. Self-organization is a process in which pattern emerges at the global (collective) level by means of interactions among components of the system at the individual level without the guidance of well-informed leaders, and without any set of predetermined blueprints, recipes or templates to explicitly specify the pattern."

Def2 ([Heyl01]):
"Self-organization is a process where the organization (constraint, redundancy) of a system spontaneously increases, i.e. without this increase being controlled by the environment or an encompassing or otherwise external system. Self-organization is basically the spontaneous creation of a globally coherent pattern out of the local interactions between initially independent components."

Def3([CDFSTB01]):
"Self-organization is a process whereby pattern at the global level of a system emerges solely from interactions among the lower-level components of the system. The rules specifying the interactions among the system's components are executed using only local information, without reference to the global pattern."

Obviously, self-organization appears in a system without interventions by external directing influences (instructions from a "supervisory leader" or an order imposed on them in many different ways – various directives, recipes, templates) and forms patterns through multiple interactions among their components. This appearance means that a functional structure appears and maintains spontaneously. Nevertheless, we can say that the complex system is not arbitrarily regulated. It is ordered in a very organized way. This organization was not built into the system at its origin, it has emerged in a sequence of self-organizing processes that understood spontaneous transitions into new states of higher organizational complexity. The term "spontaneous" doesn't mean "they just happen" for no particular reason. Patterns are well organized structures ([CaDFSTB03]) and can refer to an arrangement of objects both in space (e.g., a zebra's coat) and in time (e.g., firefly flashing). The challenge and need is to find some principles additional to the low level laws to explain it. Self-organization could be seen as the evolution of order from a disordered start ([Roch98]). According to ([Gold97]), a self-organizing system possesses multiple interdependent components that cooperate in self-initiated interactions. Through their synergy and internal interaction, a necessary information exchange is done. Such a type of system expresses a certain level of self-configuration (i.e., it is capable to construct itself through the arrangement of its constituent parts) and self- maintenance (i.e., adaptation to change). As the environmental changes are constant, it constantly adapts its behaviour.

([Roch98]) explains self-organization through the notions of eigenvalues and eigenbehaviour. He refers to the notion of *eigenbehavior* as the ability of an organization to classify its environment, and defines *eigenvalues* as the existence of some stable structures.

Def4 ([Roch98]):

> "Eigenvalues are discrete representations of observables maintained by the successive cognitive operations of a cognitive agent. An eigenvalue of an organizationally closed system can be seen as an attractor of a self-organizing dynamical system."

An *attractor* usually refers to a preferred position for the system. The type of system of interest, i.e., dynamic complex system may have many possible attractors. A system changes its state, from state s_n to state s_m, and the previous one (s_n) is called a pre-image of the next one (s_m). It is on the trajectory that leads into state s_m. The first possible pre-image (that itself has no pre-image) is the starting point for a trajectory.

A *state space* (or phase space) is a set of all possible combinations (of states) available to the system. As the possible states grow rapidly with complexity, dynamical systems possess very large state spaces. If some initial conditions are introduced, such systems typically converge to small areas of the state space (so-called attractor basins) which can be interpreted as a form of self-organization ([Heyl01]). Examples of such structures created by self-organizing systems are ant paths. They can be viewed as stationary states of a dynamical adaptive system that are stable as long as the conditions under which they were created are stable, but when the conditions change, the equilibrium automatically adjusts itself to a new stationary state. One of the properties of a self-organizing system is the possibility to re-establish the stationary state, i.e., to self-repair, if the structure of the system is damaged in some way. During these changes of state, a system converges to the attractor ([Roch98]).

Self-organization in a system reflects at different levels (from the lowest level to the highest one), and each of these levels can exhibit their own self-organization. A self-organizing system consists of a large number of interacting components that are constantly changing their state. "Decisions" and consequently changes are local (e.g. in an ant colony, each ant "decides" by its own which path it will choose). Also, components only interact with their immediate "neighbours". Mutual dependency implies that changes are not arbitrary: some relative states are "preferable", in the sense that they will be reinforced or stabilized (like those paths where there are more pheromone in nature swarm intelligence), while others are eliminated. The components of the lowest level produce their own emergent properties (patterns) and form the building blocks for the next higher level of organization, with different emergent properties, and this process can proceed to higher levels in turn.

The interesting property of self-organizing system is the interaction of components between different levels, while self-organization already exists on each particular level. These can in turn self-organize into even higher level components, i.e., self-organization between different levels can occur.

Most of dynamic systems are metastable, i.e., possessing many attractors as alternative stable positions. The role of "noise", i.e., fluctuations in such dynamic system is, therefore, very important as it allows the system to escape one basin and to enter another, leading the system (over time) in approaching of an optimum organization. The basic mechanism underlying self-organization is the deterministic or stochastic variation that governs any dynamic system. This variation allows for exploring of different regions in a state space until it happens to reach an attractor. The exploration of a state space can be emphasized, accelerated and deepened by increasing variation, i.e., by adding "noise" to the system. Reaching the attractor, the system comes to the stable state. In order to continue an exploration of new state space positions, random changes are necessary to be introduced. It can cause the system to move towards a new attractor, which forms the self-organized state ([Heyl01]). Mathematically speaking, it is possible to have several local optima, but only one global optimum.

The self-organizational mechanisms have a fully distributed characteristic in a dynamical system, i.e., it must be distributed over all participating components. An opposite situation where the mechanism is centralized in a subsystem or module will lead to the possibility that this module could be removed and the system would lose its organization.

In the following, broad *principles/characteristics* of self-organization are identified ([Macl04]). Up to this point (i.e., from the definition and description of self-organization), the following features of self-organization can be noticed: absence of external control (autonomy), dynamic operations, multiple equilibriums (many possible attractors), distributed "control", hierarchies (multiple nested self-organized levels). However, the typical characteristics also include the following issues ([Heyl01]):

<u>Global order from local interactions</u>

([Macl04]) defined one of the central principles of self-organization: complex, adaptive macrobehavior emerges from simple, local microdecisions.

Robustness, resilience

Self-organized systems function with a goal to preserve its own maintenance, and therefore are robust and capable to resist perturbations, errors as well as a partial destruction. This robustness is achieved by distributed control so that damage can be restored by the remaining, undamaged sections and a system can get back to its initial state. Random perturbations are connected to fluctuation in the system and could even help system in achieving an ever better organization ([Macl04]).

Non-linearity and feedback

Positive and negative feedback do not mean desirability. The negative feedback loop tends to slow down a process, to bring a process to equilibrium, to stabilize the system, while the positive feedback loop tends to speed it up, leading to instability, to accelerate it away from equilibrium. It biases exploration into directions, so that the system can begin exploiting information before it has finished gathering it ([Heyl01]). Their interaction represents an adaptive balance between exploration and exploitation. Feedback implies nonlinearity of a system.

Organizational closure, hierarchy and emergence

The correlation between separate components defines an ordered configuration, but not yet the organization that can be defined as the characteristic of being ordered (or structured) so as to fulfil a particular function ([Macl04]). This function is the maintenance of a particular configuration, in spite of disturbances. This general characteristic refers to the concept of closure. More generally, a self-organizing system may be divided into a number of relatively autonomous, organizationally closed subsystems that interact in an indirect way. It can be seen as a hierarchical, "boxes within boxes" architecture, where a number of relatively autonomous, closed organizations can be distinguished at each level. The organizational closure turns a collection of interacting elements into an individual, coherent whole. This whole has properties, so-called *emergence* that arises out of its organization. This is an appearance of a higher level property or feature not previously seen as a functional characteristic of a system, i.e., qualitatively new pattern and structure. It arises unexpectedly ([Heyl01]).

Bifurcations, symmetry breaking

The feature of non-linearity implies that there is a range of stable configurations in which the system may settle and that depends on a chance fluctuation. Although the individual components all behave differently, on the global, macroscopic level, the system is homogeneous and *symmetric* (from each direction observed, it will look the same). Self-organization means a searching for the best current state and therefore, among initially all equal configurations, only one possibility has a preference. Thus, one direction or one configuration dominates all others, and therefore the symmetry is lost. For such a choice - to achieve a preferred stable configuration, there are no objective criteria as the system makes an arbitrary decision ([Heyl01]).

Bifurcation describes a process caused by possibly a small change in one parameter that results in a system splitting into two possible behaviours. Further changes to the parameter then cause further splits at regular intervals until finally the system enters a chaotic phase ([Heyl01]).

Stigmergy

Stigmergy is a mechanism of indirect coordination and reciprocal relationships between components (workers, agents) of a system and the structures that they build. The term was coined in ([BTDAC97]) and it describes the following principle: the trace left in the environment by an action stimulates the performance of subsequent actions that tend to reinforce and build on each other, leading to the spontaneous emergence of coherent pattern. Therefore there is no need for an external blueprint or project leader. Each agent "encountering the project" knows exactly what it needs to do, e.g., in ant population, each ant "knows" what is its task. Stigmergy permits the use of simpler agents and decreases direct communication between agents. For example, looking again to the ant population, ant can be seen as a simple agent and a communication between them proceeds through a different amount of pheromone, laid to the paths. There are two different kinds of stigmergy ([Heyl01]): quantitative (continuous), where quantitatively different stimuli trigger quantitatively different behaviours and qualitative (discrete), where stimuli are classified into distinct categories, which trigger distinct behaviours.

Circular Causality

This principle is also known as the macro/micro feedback loop, which means that global order emerges from the interaction of the agents, and they in turn respond to the global order.

Formal Description

([Heyl01]) presented some formal concepts of a state space, an attractor and fitness landscapes.

Def 5([Heyl01]):

"A *state space* of a system is the set of all possible states of that system, and defined as set with finite, discrete or continuous number of elements: $S = \{s_1, s_2, s_3, ...\}$. For simplicity, we will assume that the state space is discrete and finite (it can be gener-

> alized to the continuous case). If a system A consists of n different subsystems or components A^1, A^2, A^3, ..., A^n that can vary independently, then A's overall state space $S(A)$ is the Cartesian product of the state spaces of its components:
>
> $$S = S^1 \times S^2 \times ... S^n, s \in S = (s^1, s^2, ..., s^n)$$
>
> The dimension of S is the product of the dimensions of all of the component spaces."

Self-organizing systems usually consist of a huge number of components and therefore, can only be modelled by statistical methods, i.e., by calculating the probability $P(s)$ that the system is in a particular state s, given a limited number of properties that have been determined by observation. The function P: $S \rightarrow [0, 1]$ assignes a probability to each state and determines a probability distribution over the state space ([Heyl01]). In order to introduce a definition of an attractor, some additional explanations are provided.

The function that describes how the system moves from one state to another in the course of time t is needed in order to model the evolution of a system. Such function f_T: $S \rightarrow S$, $f_T(s_1) = s_2$, where s_1 is the state of a system in time t and s_2 is the state of a system in time $t+T$, is usually the solution of a differential or difference equation. In principle, self-organizing systems dissipate energy, thus dissipated energy cannot be recovered in order to undo the process. This implies that the evolution of complex systems is irreversible, i.e., a past state is impossible to reconstruct from the present state.

The appropriate stochastic process can be modelled as a Markov chain ([Heyl01]): for each initial state s_i of a system, it gives the probability of a transition to a next state s_j: $P(s_j|s_i) = \boldsymbol{M}_{ij} \in [0, 1]$ where $\boldsymbol{M}$ is the transition matrix of the Markov chain. If a probability distribution for the initial state is $P(s_i,t)$, then the probability distribution for the next state:

$$P(s_j, t+1) = \sum_i M_{ij} P(s_i, t) \qquad (2.1)$$

The attractor of a system is defined in a following way.

Def 6 ([Heyl01]):

> "An *attractor* is a subset A of the system's state space S that the system can enter but not leave, and which contains no smaller subset with the same property. This means that:
>
> $$(\forall\, s_i \in A)\ (\forall\, s_j \notin A)\ (\forall\, n, T)\ f_T(s_i) \in A \vee M^n_{ij} = 0 \qquad (2.2)$$

The property of not containing a smaller such set can be expressed as:

$$\lim_{n \to \infty} M_{ik}^{n} = 0 \Leftrightarrow s_k \notin A \qquad (2.3)."$$

The previous definition describes causal closure, i.e., inside the attractor, the process has "closed in" on itself. There are many different shapes, sizes and dimensions of attractors. Some of them are:

2.2 A *zero*-dimensional point attractor consists of a single state and this is the situation where a system reaches equilibrium.

2.3 In *one*-dimensional limit cycle, all states of the attractor are revisited at regular intervals and it represents far-from-equilibrium configurations where the system exhibits periodical behaviour.

2.4 A *non-integer*, *fractal* dimension is a characteristic of a so-called "strange" attractor and it is connected to certain chaotic processes.

Def 7 ([Heyl01]):

"A *basin* $B(A)$ of an attractor is a set of states outside a given attractor whose evolution necessarily ends up inside:

$$(\forall\ s \in B(A))\ s \notin A \wedge (\exists\ T)\ f_T(s) \in A \qquad (2.4)$$

In a deterministic system, every state either belongs to an attractor or to an attractor basin. In a stochastic system, there is a third category of states that can end up in either of several attractors."

The complex structure of attractors and basins can be replaced by the more intuitive model of a fitness landscape which is explained in subsection 2.2.1.

Although there are not many practical applications of artificial self-organizing systems, such systems offer several advantages over more traditional systems: robustness, flexibility, capability to function autonomously while demanding a minimum of supervision, and the spontaneous development of complex adaptations without need for detailed planning. Disadvantages are limited predictability and difficulty of control. Nevertheless, the basic mechanisms underlying self-organization are not yet clear enough.

2.1.2 Nature Based Mechanisms

The most popular self-* approaches are those ones that scoop their power from different bio-inspired mechanisms. Based on these mechanisms, a variety of swarm intelligent algorithms and adaptive algorithms are constructed (ant intelligence, bee intelligence, immune system behaviour, hormone system behaviour, evolutionary and genetic algorithms). They have been applied to different problems of optimization, searching and routing, e.g.: travelling salesman problem ([DoSt05]; [WoLC08]; [Potv96]), job-shop scheduling problem ([CSLG06]), scheduling workflow applications in cloud computing environments ([PWGB10]), vehicle routing problem ([ToVi02]), assignment problem ([Stüt97); set problem ([HRTB00]), network routing ([DiDo98a]), data mining ([PaLF02]), grid workflow scheduling problem ([ChZh09]), image processing ([NeSR06]), power electronic circuit design ([ZhLC06]), the Internet server optimization problem ([NaTo04]), document clustering ([TaVe05]), etc.

Ant Algorithms

The ant colony optimization (ACO) metaheuristic has been inspired by biological (swarm) systems – the real ant colonies. Ants' behavior is characterized by an indirect communication between individuals in a colony via pheromone. Mapping to the artificial ant colony where a software agent plays the role of an ant, ACO means multi-agent organization. The natural pheromone is stigmergic information that serves for the communication among the agents. Ants make pure local decisions and work in a fully distributed way. In ACO, ants construct solutions by moving from the origin to the destination, step-by-step, according to a stochastic decision policy. After that, the aim of pheromone update is to increase the pheromone values associated with good solutions (deposit pheromones) and decrease those associated with bad ones. More details can be found in ([DoSt05]).

The most popular variations and extensions of ACO algorithms are ([DoSt05]): Elitist Ant System, Rank-Based Ant System, Min-Max Ant System (MMAS), and Ant Colony System. ACO algorithms have been applied to different types of problems ([DoSt09]), e.g., (network) routing, assignment, scheduling, and machine learning. ACO can be combined with other algorithms, e.g., genetic algorithms ([RoMe08]) forming hybrid algorithms. AntNet ([DiDo98a]; [DiDo98b]) is a network routing algorithm based on ACO. It is an algorithm for adaptive routing in IP networks, it is highly adaptive to network and traffic changes, robust to agent failures and provides multipath routing. AntNet algorithm supports high dynamics of joining and leaving nodes. A detailed description of these algorithms can be found in ([DiDo98a], [DiDo98b]) and ([DoSt05]).

Ant algorithms fit mostly to problems that treat optimization, searching, and some adapted ant algorithms are suitable also for clustering. Ant algorithms have a good theoretical base ([DoSt05]).

Bee Algorithms

Bee Algorithms are a relatively new and promising approach with just a few applications up to now. The biological background of bee behaviour is characterized by autonomy and distributed functioning, and self-organization ([CaSn91]). A honeybee colony of one hive contains bees with different roles: foragers, followers, and receivers. Self-* of bees relies on two main strategies, navigation and recruitment. The navigation strategy is concerned with searching for nectar of flowers in an unknown landscape. A forager scouts for a flower with good nectar and after finding and collecting it returns to the hive and unloads the nectar. Afterwards, the forager performs a recruitment strategy (a so-called "waggle dance"), meaning that it communicates the knowledge about the visited flowers to other bees. This serves to inform other members of the hive about the quality, distance and direction of the found nectar. A follower chooses to follow a forager at random and visit the flower that has been "advertised". It does not need a decision about navigation on its own and therefore, is more efficient. A forager can choose to become a follower in the next step of navigation, if it observes better information about nectar (through the recruitment process of some other forager), i.e., foragers and followers can change their roles. A receiver always stays in the hive (stationary) and processes the nectar.

Bee-inspired algorithms have been applied so far mainly in searching and optimization like travelling salesman problem ([WoLC08]), job shop scheduling ([CSLG06]), routing and wave-length assignment in all-optical networks ([MaTA07]). We adapted bee intelligence for the realization of load-balancing policies in the load-balancing problem and implemented a load-balancing bee algorithm ([ŠeKü08]).

Bee algorithms still have no established theoretical base.

Other bio-inspired mechanisms

A lot of research work exists dedicated to the mapping of naturally based mechanisms (e.g., evolutionary algorithms, neural networks) as well as some forms of swarm intelligence mechanisms (e.g., like ant algorithms, bee algorithms etc.). The following mechanisms are also inspired by nature.

([KTTRS09]) use the coordination mechanism of slime mold for wireless sensor and actor networks. They adapted the tubular network formation behavior of slime mold to design coordination protocols for wireless sensor and actor networks. ([ScCr07]) use a technique of signal propagation that was inspired by slime mold and applied it to a navigation principle for swarm robotics. Using the slime mold-inspired strategy, the simulated robots successfully perform a collective cleaning scenario and show the ability of finding the shortest path between two target places. ([MoMa08]) uses slime mold as a model for numerical single objective optimization. The artificial fish schooling algorithm is articulated and described ([Farz09]; [BLLNL09]). Still there are enough places to develop/adapt it further towards to wider classes of problems. ([TyAB06]) uses fireflies as role models for synchronization in ad hoc networks. They review fireflies' synchronization process by looking at experiments that were made on fireflies and the mathematical model of ([MiSt90]), which provides key rules to obtaining a synchronized network in a decentralized manner. This model is applied to wireless ad hoc networks. ([WTPWN05]) have been inspired by fireflies in the sensor networks synchronicity. They present the Reach-back Firefly Algorithm (RFA), a decentralized synchronicity algorithm based on a mathematical model that describes how fireflies and neurons spontaneously synchronize. ([CuWa09]) apply firefly's synchronicity to wireless sensor networks.

Genetic Algorithms

Genetic Algorithms (GAs) ([BeBM93a], [BeBM93b], [Gold89]) are a kind of mathematical simulation for Darwin's theory of evolution. The starting point is a formation of an initial population either using some particular method or at random. The elements of an initial population, individuals, are points from the searching space for a given problem. Every individual is uniquely determined by its genetic material. The adaptation of every individual has to be found according to the fact how good solution that individual is. Therefore, the appropriate value of the fitness function is assigned to each individual. Using a selection operator and the values of the fitness function, "the best fitted" individuals are being chosen. A new population is formed by using crossover and mutation operators. The rules of genetic and evolution implies a greater probability that a new generation have a better genetic material. Iterating this procedure from generation to generation, the genetic material of the population becomes better and the process should converge to the optimal solution of the given problem. If the specified number of generations is reached or some criteria of convergence are fulfilled, the procedure stops.

GAs have been applied to a huge number of problems that treat searching and optimization, e.g. just to mention a few of them: numerical and combinatorial optimization problems like travelling salesman problem ([Potv96], [KLŠF98]), circuit design ([XuDH09]), job shop scheduling ([Davi85]), machine learning ([Gold89]), economic models ([Dawi96]), geophysical inverse problem ([ŠeKr99], [ŠeKT00], [ŠeTo02]), uncapacitated warehouse location problem ([KFŠT96]), the Internet search ([ŠeCv02]), improving the execution time of the algorithm itself through some theoretical consideration ([KrRŠ97]), etc.

Immune System Based Algorithms

The Artificial Immune System Algorithm (AIS) ([Yu08], [LiDW09], [TaVe05]) is inspired by processes in immunology. In basic AIS, the starting point is the encoding of "antibodies". Typically, an antigen is the target, e.g. the data item we need to check to see if it is an intrusion. The antibodies are the remaining targets in the database. Sometimes, there can be more than one antigen at a time, and there are usually a large number of antibodies present simultaneously. Antigens and antibodies are represented in the same way. The next step is to determine the similarity measure or matching rule. This is one of the most important design choices in developing an AIS algorithm, and is closely coupled to the encoding scheme. Finally, a selection is performed in every generation based on the affinity of antibodies. The expected number of times an antibody is selected is proportional to its affinity. The selected antibodies replace the existing population and form the next generation of population. Mutation is performed on all selected antibodies.

AISs are used in several applications such as anomaly detection ([DaFo96]), pattern recognition ([CaDa03]), data mining ([TiNK02]), computer security ([HoFo00]), adaptive control ([KrNe99]) and fault detection ([BrTy00]). A theoretical description of AIS can be found in ([TaDa00]).

Hormone System Based Algorithms

Artificial Hormone System (AHS) uses messages like hormones use the blood circuit in the human body ([TrTU06]). ([TrTU06]) proposes an AHS that consists of four parts that can be directly mapped to human counterparts. First, the function of the cells is compared and measured by metrics to calculate a reaction. Second, the hormone producing tissues of the human body are influenced by receptors which observe the environment to trigger hormone production. This behaviour is mapped by monitors that collect information locally about the running services and transfer back this information onto outgoing messages. Third, the function of the cells receptors is mapped to monitors for incoming messages in order to collect the information transferred back and hand them over to the metrics. Fourth, the digital hormone values are carrying the information. To further reduce the amount of information needed to exchange, ([TrTU06]) assumes that the digital hormone value enfolds both, the activator as well as the inhibitor hormone. If the value of the digital hormone is above a given level, it activates while a lower value inhibits the reaction.

AHS has been used in several applications like construction of "organic" middleware ([BrRP08]), self-organization of network nodes ([TrTU06]), and examination of task distribution ([VoBW08]). According to our investigations, algorithms and formalization of AHS are still open research issues.

Obviously, nature is prolific with a variety of self-* properties and mechanisms that are attractive enough to be a challenge for IT scientists in order to map them into the scope of IT complexity problem. Except adaptive algorithms (from which genetics algorithms are used the most) and one type of swarm algorithms (ant intelligence), the numbered bio-mechanisms are almost totally or partially unexploited, leaving the place for new researches.

2.1.3 Socially Based Mechanisms

Self–organizing mechanisms can be found also in human society (sociology and sociodynamics, economics, behavioral finance, and anthropology). Examples from *sociology* and *sociodynamics* are described in the following text.

Critical mass

This phenomenon, investigated and described in sociophysics, refers to the existence of "sufficient momentum" in a social system such that it becomes self-sustaining and fuels further growth. The mechanism can be classified as a kind of self-aggregation and the idea is to capture it in a mathematical model although this may not be possible for the behaviour of any particular individual ([Ball04]).

Herd behaviour

This notion in modern psychological and economic science denotes a special behavior of humans ("inherited" from the animal society) where large numbers of people are acting in the same way at the same time. As an example of a benign herding behavior, the following situation can be described. A family needs to decide which of two restaurants to choose. The decision is difficult as both restaurants look appealing from one side, but both are empty from the other side. So, a family decides randomly and chooses restaurant R_1. After a while, a man walks down the same street wishing to find a good restaurant. He sees that restaurant R_1 has customers, while R_2 is empty, and therefore, he concludes having customers makes it the better choice and chooses R_1. This phenomenon is also referred as an information cascade ([Bane92], [BiHW92]) and it typically occurs when people make their own decisions and choices based on the previous observations of the actions of others. As a consequence, they make the same choice like the others, independently of their own private information signals. Although it is usually assumed to be the result of rational choice, information cascades can sometimes lead to arbitrary or even erroneous decisions.

In principle, this phenomenon describes an emerging of new models of behaviour in a different situation through the behaviour of individuals in a group that act together without planned direction. A crowd differs significantly from the behaviour and psychology of those individuals within it. Usually, these are situations that leave little time for decision making ([Berk74]).

Groupthink

It is a type of thought-behavior within a cohesive in-group whose members try to minimize conflict and reach consensus. Despite some characteristics of self-organization, this mechanism is exposed to criticism due to the fact that such a consensus is usually reached without critically testing, analyzing and evaluating ideas. Also, individual creativity, uniqueness, and independent thinking are lost in a deeply cohesive group ([BuHu97]).

Social Learning

Knowledge propagation, as an important form of behaviour in a social learning dynamic, leads to cognitive development. This well-know mechanism from a human society was applied in technology showing promising results ([Vygo78]).

In *economics*, a market economy has some self-organizing mechanisms. ([Krug96]) states that market self-organization plays role in the business cycle. In business and economic systems, the individual behaviour's primary goal is to increase the profit. Dynamics of a system is handled by the activity developed to face business and economic constraints. Examples from economics and business include market-based mechanisms and business related mechanisms.

Market-based mechanisms ([FoCM02], [Brunn01]) that are inherited from economic markets model systems in correlation to some economic model. Participating individuals act towards increasing their personal profit or utility. The parameters of a system depict macroeconomic variables (e.g., economic growth), whereas the parameters of the individuals correspond to microeconomic parameters. An example of economic-based self-mechanism is creative destruction.

Creative Destruction

It refers to a situation when a new setting eliminates an old one leading to economic development. For example, an economic system must destroy less efficient firms in order to make room for new, possibly more efficient entrants ([Schu02]).

Business related mechanisms ([Stew01]) are based on business models and theories which use self-organisation. A new business environment faces unpredictable behaviours and fast changes. Consequently, contemporary business models shifted from efficiency to flexibility and the speed of adaptation. Therefore, the focus in such models is on the complex relationships between different business components. The examples of business related mechanisms are personalized marketing and activity-based management.

Personalized Marketing

This strategy is adjusted for each individual customer and evolved according to customer reactions. An example is the one-to-one variable pricing model, i.e., providing an individual offer to each customer using Internet technologies ([WaYW04], [HaGr02]). One for of personalised marketing is syndication: the sale of the same good to many customers, who then integrate it with other offerings and redistribute it ([Werb00]).

Activity-based Management

In this strategy, networks of working groups can change their structure, links and behaviour in response to business requirements ([ViSk02]) in order to capture the self-organisation decisions that need to be taken during the business operations. The objective is to solve potential conflicts of interests in both the inner and the external co-operative activity of the company. It is assumed that the structure of the company is virtual without clear hierarchy and control. The mechanism forces effects that can be initiated both vertically and horizontally via "round table meetings", which could be meetings normally held in companies to assess results and handle exceptions.

([ZoJM06]) investigate a multi-agent model based on the formalization of psychological and organizational theories. Multi-agent simulation is applied to explore social self-organization when people have to perform a task.

([HDKC06]) discusses examples of socially inspired self-organization approaches and their use to build socially-aware, self-organizing computing systems. They present different mechanisms originating from existing social systems: stigmergy from social insects' behaviors, epidemic spreading, gossiping, trust and reputation inspired by human social behaviors, as well as other approaches from social science related to business and economics. The emphasis is put on social network dynamics, social network patterns, social networks analysis, and their relation to the process of self-organization. The applicability of socially inspired approaches in the engineering of self-organizing computing systems was illustrated with applications concerning WWW, computer networks and business communities.

([BaBD02]) based their work on experiences-learning network about sustainable work systems with six companies and two research institutes. They describe the intra-organizational conditions and social complexity in an effort to establish realistic alternatives for the work organization that support and develop sustainability for the organization and its personnel. They observe and identify self-mechanisms and organizational levels in a company through the complexity of a business life, diagnose self-organizing mechanisms, explain management and leadership in self-organizing process, and introduce the notion of emergence in the scope of business life.

([Fuch03]) investigated human co-operation and diagnosed emergent properties on upper hierarchical systemic levels. Co-operation means co-action and takes place permanently in re-creative systems: two or more actors act together in a coordinated manner, producing a new emergent property.

Although self-organization in socio-domain exists as well as socially based mechanisms, their application is unexploited enough (the related work from this area is presented above). Comparing with a number of self-* properties from nature, this domain is "poorer", however it deserves attention.

2.1.4 Self-Organization in P2P

P2P Systems

Distributed computer architectures labelled peer-to-peer (P2P) have grown dramatically in recent years, increasingly becoming popular because they offer opportunities for real-time communication, ad-hoc collaboration and information sharing in a large-scale distributed environment ([AnSp04]). P2P systems share computer resources (content, storage, CPU cycles) and information through direct exchange and allow for a symmetric communication between the peers; by means that each peer has both a client and a server role. One of the most important characteristics of P2P systems is their ability to adapt to failures (of nodes or connections between nodes) and to continue functioning without an interruption. They are capable to self-organize, i.e. to react to changes without the need of a central server. There are many definitions of P2P systems. One definition, proposed in ([AnSp04]) is:

"Peer-to-peer systems are distributed systems consisting of interconnected nodes able to self-organize into network topologies with the purpose of sharing resources such as content, CPU cycles, storage and bandwidth, capable of adapting to failures and accommodating transient populations of nodes while maintaining acceptable connectivity and performance, without requiring the intermediation or support of a global centralized server or authority."

P2P systems can be classified into three categories ([AnSp04]): communication and collaboration, distributed computation, and content distribution. Usually the terms "node", "peer" and "user" are used interchangeably, according to the context, to refer to the entities that are connected in a P2P network.

P2P architectures have been employed for a variety of different application categories ([AnSp04]): communication and collaboration, distributed computation Internet service support, database systems content distribution.

Nowadays, P2P content distribution systems are maybe the most interesting category. Each operation in any P2P content distribution system goes through a network of peer computers (nodes), and connections (edges) between them. This network is virtual one, formed on top of one (or more) existing networks. It is independent on the underlying physical computer (typically IP) network, and is referred to as an "overlay" network. The operation of the P2P system and its functioning strongly depends on the topology, structure, and degree of centralization of the overlay network as well as the routing and location mechanisms it employs. P2P overlay networks can be ([AnSp04]):

- Purely Decentralized Architectures: all nodes in the network are equal; each node has both a client and a server role; there is no central coordination of their activities;

- Partially Centralized Architectures. some of the nodes have a more important role (so-called "supernodes"); they are dynamically assigned and act as local central indexes for files shared by local peers; therefore, they are not single points of failure for a peer-to-peer network, since if they fail, the network will automatically take action to replace them with others;
- Hybrid Decentralized Architectures: a central server exists and it is responsible for maintaining and facilitating the interaction between peers; the central server is a single point of failure (the central server); this could imply a potentially vulnerable system, subjected to technical failures or malicious attacks.

Overlay networks can be classified into two main categories according to their structure:

- Unstructured: content typically needs to be located as the placement of content is completely unrelated to the overlay topology. Searching mechanisms expose emergent phenomena driven from purely local interactions and range from brute force methods to more sophisticated methods (flooding the network with propagating queries, the use of random walks and routing indices). Examples of unstructured systems are Napster, Publius [WaRC00], Gnutella [Gnut03], Kazaa [Kaza03], Edutella [NWQDS03], FreeHaven [DiFM00].
- Structured: a globally consistent protocol is used to ensure that any node can efficiently route a search to some peer that has the desired file. The overlay topology is tightly controlled and files (or pointers to them) are placed at precisely specified locations. A mapping between content and location is typically provided in the form of a distributed routing table. The most common type of structured P2P network is the distributed hash table. Examples of structured systems are Chord [SMKKB01], CAN [RFHK01], PAST [DrRo01], Tapestry [ZhKJ01].
- A category of networks that are in between structured and unstructured are referred to as loosely structured networks. A typical example is Freenet ([ClSW00], [CHSW02]).

The most important "pros and cons" in "unstructured P2P networks versus structured P2P networks" concern the issues of scalability and support of dynamics. Namely, a structured P2P network scales well, but the support of dynamics is not so good. The queries can be only "exact match queries" instead of more complex ones. In contrast, in an unstructured P2P network, a query must be more "sophisticated", a placement of information can be done independently of an overlay topology, but the content must be localized explicitly. A disadvantage is that it does not scale well. However, it is very well suitable for dynamic populations.

Examples for unstructured P2P networks are ([AnSp04]): hybrid decentralized (Napster and Publius systems), purely decentralized (Gnutella, Free-Haven and partially centralized (Kazaa, Edutella).

Examples for structured P2P networks are ([AnSp04]): Freenet, Chord, CAN, Tapestry, Pastry, Kademlia ([MaBi02]).

Application of Self-Organization in P2P Systems

A certain amount of applications of self-organization can be found in P2P systems. In most cases, work in this area leans to the self-* properties of P2P themselves and eventually, to some specific algorithms used in particular overlay networks.

([ACDDHSP03]) describes the PGrid system that combines the best characteristic of both unstructured and structured P2P systems by using self-organization principles for constructing and maintaining a DHT-like routing infrastructure. It takes advantage of the resulting emergent properties for improving various services including routing, updates and identity management.

([MeKo05]) investigates which features of self-organization exist in P2P (like decentralization for resource mediation and access, etc). The paper puts in question whether non-linearity would be attractive for peer-to-peer systems and to what extend. Also, it considers how self-organization will fit together with some issues, which need much more control and management mechanisms (security, anonymity).

([POAKS09]) discuss difficulties to achieve a high level of Quality-of-Service for P2P traffic due to dynamic nature of P2P systems. The paper classifies relevant self-organizing aspects of P2P systems, in order to understand better such self-mechanisms and how they affect the underlying Internet infrastructure.

([LTSS02]) addresses the problem of forming groups in peer-to-peer (P2P) systems and examines what dependability means in decentralized distributed systems. The authors argue about how this global state remains stable as nodes enter and leave the system. They introduce a self-organizing hierarchically-based P2P system and evaluate the reliability of Chord versus the hierarchical grouping system through simulation experiments. According to their obtained results, both systems perform adequately in a range of tolerance for failure under normal conditions as well as they utilize self-configuration.

([BaJe08]) overviews bio-inspired techniques in order to implement self-properties in large-scale, decentralized networks, and proposes gossip-based algorithms. The paper describes a decentralized approach to arrange large numbers of mobile agents into different formations. The approach is inspired by the biological mechanism of cell sorting via differential adhesion.

([BBJM07]) states that the major challenge is represented by the dynamic character of P2P overlay networks, the unreliable communication channels and the lack of reliable and robust components. The paper presents a heartbeat synchronization protocol for overlay networks inspired by mathematical models of flash synchronization in certain species of fireflies. In their protocol, nodes send flash messages to their neighbours when a local heartbeat triggers. They adjust the phase of their next heartbeat based on incoming flash messages using an algorithm inspired by mathematical models of firefly synchronization.

([JoLi10]) investigates a hybrid use of both paradigms (unstructured and structured). Their work is based on a fully decentralized algorithm to build such hybrid systems, as existing methods often require human intervention and some centralized gateway to select peers and guide them to build the structured overlay.

2.1.5 Self-Organization in SBC

Coordination Models

The distributed shared memory paradigm ([Kühn01]) for coordination between parallel processes has been relatively often used in the last two decades. A rapid development of parallel and distributed systems and massive usage of huge number of processors imposed the challenge: coordination of the cooperation among very large numbers of active entities. This implied design and implementation of a number of coordination models and their associated programming languages. A number of existing coordination models and their associated programming languages contribute successfully in parallel systems' applications. The coordination paradigm possesses many advantages, e.g., it hides the complexity of communication by offering the abstraction of reading from and writing data into a virtually shared space, allows processes computing in different languages and platforms to interoperate using the same primitives, provides a number of features, helpful to build distributed operational processes ([PaAr98]). There are many definition of the notion of coordination. One way to define coordination is the following.

Def1([CaGe89]):
"Coordination is the process of building programs by gluing together active pieces."

([SaCM99]) argued that a clear separation of computation, communication and coordination reduces the complexity of system design and provides for more stable and reliable system implementations.

Coordination models and languages advocate a distinct separation between the internal behaviour of the entities and their interaction. The purpose of a coordination model and associated language is to allow for integrating a number of heterogeneous components.

Coordination models and languages are classified into two groups: data-driven and control-driven. Each category is suitable for a different type of application domain: the data-driven category is used mostly for parallelising computational problems, whereas the control-driven category is suitable for usage in modelling systems ([PaAr98]).

Data-driven Coordination. In data-driven coordination models, coordination is achieved by exchanging coordination information via shared data. The coordination component is usually a set of primitives with predefined functionality which is used in connection with some "host" computational language. This type of coordination usually refers to the existence of a kind of a mixture of coordination and computation code within a process definition.

In data-driven coordination language, the processes cannot easily be distinguished as either coordination or computational processes, because coordination primitives are mixes with a computational part. It depends on a program designer to clearly separate the coordination and the computation.

Control-driven coordination. In the control-driven coordination models, there is a complete separation of coordination components from computational components that are treated as black boxes with clearly defined input/output interfaces. The coordinated systems do not influence the coordination process directly and thus, play a passive role.

Linda Coordination Model

The original coordination model Linda ([Gele85]) was developed by David Gelernter and Nicholas Carriero and used to coordinate the computations among several parallel processes. It operates with a logically shared memory, called tuple space. In principal, the tuple space is an implementation of the associative memory paradigm for a distributed computing that provides a repository for tuples. From mathematical point of view, instead of a repository, it is usually called a bag. A tuple is a finite-sized ordered list of elements.

The defined operations are: *out* - writing tuple to the tuple space, *rd* - reading that matches the provided template parameters, *in* - reading and removing tuple from the tuple space, and *eval* - generating a tuple and writes it to the tuple space. The tuple values can be provided explicitly or as function parameters.

Based on the original Linda, a diversity of system implementations have been developed, e.g. JavaSpaces ([FrHA99]), T-Spaces ([WMLF98]), GigaSpaces ([Cohe10]), Lime ([PiMR99]), Rinda ([Masa09]), PyLinda ([PyLi10]), CppLinda ([Slug07]) and Prolog-Linda ([Sutc90]). The Linda model is further extended.

Space Based Computing

One style for realization of coordination paradigm is Space Based Computing (SBC). The space based computing is a powerful approach to handle the complexity of the interplay of autonomous components in heterogeneous environment through a high abstraction of the underlying hardware and operating system software. An abstract space connects distributed processing entities over a network. Processes communicate and coordinate themselves by reading and writing data structures (in a shared space). Among many useful properties as a/synchronous communication, built-in replication for fault tolerance, near-time provision of information, etc., we can say that the main advantages of this model are high decoupling and reliable communication.

"The space based computing approach is an easy to use solution that handles the complexity of the interplay of autonomous components in a heterogeneous environment through a high abstraction of the underlying hardware and operating system software" [Kühn94].

The disadvantages are: the availability of a tuple space to any process implies that a tuple space is unprotected, there is no hierarchical organization of tuples, and it may not scale well with the number of tuple spaces and processes ([BHLTT09]).

XVSM is a middleware technology and one way how the SBC architectural style can be realized ([KüRJ05], [XVSM)]). It can be used in many scenarios (like distributing of the data over multiple peers, automatic data persistency, etc). XVSM generalizes Linda tuple based communication ([CaDo98], [Kühn01]) by introducing *shared containers* as the main concept for data sharing and the place where data is stored and could be shared with other peers. Multiple containers can exist in a space leading to a structuring of the coordination space. In case that the number of containers is zero, then the space is called "empty". A container can be bounded (the number of entries is limited to a number greater than zero) and unbounded (if the number of entries has reached the maximum permitted number of entries, then the container is called "full").

Entries represent data items in a shared container. They are stored and retrieved as serializable objects. A container is a subspace where the data is stored in, and that manages entries. A formal description of this model and its navigational and extensible query language can be found in ([CrKS09], [KüMS08]). Entries can be accessed by the operations *read* – returns a number of entries from a container without destroying them*, take* - similar as read, but the entries are destroyed*, write* – writes a list of entries to the space, and *delete* - removes entries like take, but does not return them.

Each container is accessible by an Internet address. A container is Internet addressable using an URI of the scheme: *xvsm://namespace/ContainerName*.

Each container possesses so-called *coordinators*, which are responsible for the observed order of entries in the container and define the exact semantics of each operation ([KMKS09]). The container has a list of obligatory (for all entries) and optional coordinators (only for specified entries), which are specified at creation time. If coordinators like Key or Vector are obligatory for a container, each write operation must specify the necessary key or index information. On the other hand, implicit coordinators may only be optional and thus only manage a subset of entries. A container possesses one or multiple coordinators that are the programmable part of a container. However, coordinators are not only programmable part of a container.[2]

Every coordinator has its specific *selector* that serves as a kind of help in providing additional information to the coordinator. Whenever an operation is performed on a container, the parameters of the operation are collected in a selector. An access to a container can be done with or without a selector. If it is used, it provides additional information to the coordinator about the desired data. For write operations, this term was replaced by "coordination data". However, "selector" parameters are still in use for queries (read, take, delete).

Important features of XVSM are also its support for blocking operations, transactions, and notifications. When the application component invokes a method, the operation is executed on the targeted container. But, instead of returning the result to the application component, the result is written into the *answer container* ([CrKS09]). An answer container is either a physical (like ordinary container) or a virtual (it is addressed the same way as a physical one, but represents a binding in the XVSM-Application API between the identifier of the answer container and a call-back method provided by the application component). In the API, the virtual container can return the result directly to the application component or invoke an asynchronous callback function.

A detailed description of XVSM can be found in ([Mord10], [XVSM], [Mozart11]).

[2] Also, aspect can be user defined. An *aspect* is an extension of the existing functionality, i.e., various pre- and post-methods on each action (e.g., preWrite and postWrite methods). The idea is to implement these methods to perform some special action that is done automatically before or after the call of one of the methods.

Application of Self-Organization in SBC

An interesting application of self-organization techniques is in the context of coordination languages and models. The synergy between self-organization and space based computing technology is promising as space based computing offers high decoupling and a blackboard based communication for the interaction of autonomous peers. Spaces have proven to be useful for communication between autonomous agents and processes. Thus, this synergy aims at developing tools (languages, models, infrastructures) to flexibly manage the interaction of components in distributed systems.

SwarmLinda ([GrMT08]) is a Linda-based system that abstracts Linda concepts in terms of swarm intelligence (particularly ant intelligence). The implementation is based on some nature mechanisms in ant colony and is created to achieve an improvement of many characteristics such as scalability and adaptiveness.

([CMVT07a]) considers one multi-agent systems based on swarms (ants). This paper was inspired from self-organization to improve scalability and the current status of tuple organization in tuple-space systems. They present a solution that organizes tuples in large networks while requiring virtually no global knowledge about the system.

([CMVT07b]) again discusses the issue about storing tuples in a way that processes benefit from the organization of tuples. It also discusses the advantages and disadvantages in the scope of achievement in this area and states that although some progress has been made, most of the proposed solutions fail to address the reverse problem. Namely, the ideal situation will be to achieve a tuple-clustering, but to avoid an over-clustering which can affect a system's robustness. In other words, the goal is to have a balance where tuples are clustered, but not totally concentrated in very few tuple spaces.

In ([ViCG07]), a collective sort based on bio-mechanisms is discussed. Autonomous agents are assigned the task of moving tuples across different tuple spaces with the goal of reaching perfect clustering: tuples of the same kind are to be collected in the same, unique tuple space. The paper describes a self-organizing solution to this problem, where each agent moves tuples according to partial observations.

([CaVi08]) propose the alternative view of self-organizing coordination, where coordination laws are probabilistic, based on local criteria, and time-reactive. This results in coordination services where global properties of interest appear by emergence. As a proof-of-concept, they created an application inspired by corpse clustering and larvae sorting in ant colonies, where a distributed tuple-space-based scenario is enhanced with adaptive tuple clustering and sorting.

In ([ViCO09]), the authors discuss the framework of self-organizing coordination. They created the TuCSoN coordination infrastructure that can be used as a general platform for enacting self-organising coordination. Testing is done on two cases: an inter-space application of adaptive tuple clustering, and an intra-space application of chemical-like coordination reactions.

Further, ([ViCa09]) is inspired by existing literature proposing nature-inspired approaches for the coordination of complex distributed systems, and proposes a mechanism to leverage exact computational modeling of chemical reactions for achieving self-organization in system coordination. They introduce the notion of biochemical tuple spaces and the appropriate model where: a tuple resembles a chemical substance, a notion of activity/pertinency value for tuples is used to model chemical concentration, coordination rules are structured as chemical reactions evolving tuple concentration over time, a tuple space resembles a single-compartment solution, and finally a network of tuple spaces resembles a tissue-like biological system.

In [TripC08], semantic clustering and self-organization in triple space is considered. The underlying work refers to the implementation of a distributed Triple Space over Triple Space kernels. The distribution strategy stores references to the triples in distributed indexes which are found efficiently over hash values. Structural metadata is introduced as an extension of the existing Triple Space ontology. However, self-organization approaches are also outlined as a future extension to optimize the distribution of triples in the Triple Space network.

Although this synergy between self-organization and space based computing technology is promising, the research works numbered in the previous text, mainly concentrate on swarm (ant) intelligence. Additionally, the point is put on improving some disadvantages of SBC by using self-organized approaches. So, the contribution of this synergy could be found in the area of coordination models. Still, the real benefits of the combination “self-organization and SBC” in a wide application domain as well as the usage of this combination in different IT scenarios are open issues.

2.2 Algorithms

The theory of algorithms is the strongest connection between mathematics and computer sciences. Generally speaking, an algorithm is a set of precisely ordered and well-defined finite sequence of steps (instructions) for solving a certain problem [BlGu03]. The instructions describe a computation that starts from an initial state, proceeds through a well-defined series of successive states, and eventually terminating in a final ending state. However, the transition from one state to the next is not necessarily deterministic. In more advanced or abstract settings, the instructions do not necessarily constitute a finite sequence (and even not necessarily a sequence). One class of algorithms with such properties are so-called nondeterministic algorithms ([Floy67]) which have one or more choice points where multiple different continuations are possible, without any specification of which one will be taken. Some algorithms, known as randomized algorithms, incorporate randomness.

There are various ways to classify algorithms, each with its own merits ([CLRS09], [Good01], [Skie08]). The algorithms could be classified by:

- implementation (recursion or iteration, serial or parallel, deterministic or non-deterministic, exact or approximate, logical, quantum),
- design (brute force, divide and conquer, linear programming, dynamic programming, search and optimization)
- fields of study and area (combinatorial algorithms, computational mathematics, computational sciences, computer sciences, information theory and signal processing, software engineering, medical algorithms)
- complexity
- computing power

The further emphasis will be put on algorithms for search and *optimization*. This category encompasses the following subcategories: combinatorial optimization algorithms, evolutionary algorithms, heuristics, dynamic programming, and stochastic optimization. This classification does not mean that these subcategories are strictly divided. On the contrary, these subcategories overlap each other. In order to precise the further narrowing in description, the focus will be on the algorithms that are commonly called metaheuristics.

2.2.1 Metaheuristics

Many combinatorial optimization problems are NP-hard, i.e., they cannot be solved (optimally) within the polynomial bounded computational time, especially when large instances needed to be solved. Hence, in such situations when the time or resources are limited, the approximate algorithms that do not try to find an optimal solution, but an approximate solution, have to be used. These algorithms are usually called heuristics. Their advantage is that they obtain near-optimal solutions in a relatively short time [Yang08]. Further, as an extension of heuristics, a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems, was developed. This new class of algorithms, the so-called metaheuristics, increases the ability of finding very high quality solutions to hard combinatorial optimization problems in a reasonable time [Yang08]. A metaheuristic is formally defined as an iterative generation process which guides a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space, learning strategies are used to structure information in order to find efficiently near-optimal solutions [OsLa96]. The fundamental properties of metaheuristics are [BlRo03]:

• Metaheuristics are strategies that "guide" the search process.

• The goal is to efficiently explore the search space in order to find near-optimal solutions.

• Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.

• Metaheuristic algorithms are approximate and usually non-deterministic.

• They may incorporate mechanisms to avoid getting trapped in confined areas of the search space.

• The basic concepts of metaheuristics permit an abstract level description.

• Metaheuristics are not problem-specific.

• Metaheuristics may make use of domain-specific knowledge in the form of heuristics that are controlled by the upper level strategy.

• Today's more advanced metaheuristics use search experience (embodied in some form of memory) to guide the search.

The evolutionary algorithms and swarm intelligence algorithms belong to this group. To emphasize their very specific characteristics: adaptive - they change their behaviour based on the resources available, and intelligent - i.e., they are based on some form of the existing intelligence, usually from nature and map it by forming an equivalent artificial intelligence.

1) They usually manipulate with a *population of individuals*, and the solution is eventually found through many iterations of the considered population. A mapping is done by using a certain mathematical model.

2) The usefulness and fittest of a particular individual in the population is "checked" by means of a measure of the solution optimality – a *fitness function*. This is a mathematical description of the algorithm's goal and dynamics of a system. Generally speaking, a fitness function can be seen as a function f on the state space, f: $S \rightarrow \mathbf{R}$, $f = f(s)$. Every point in the space has a certain "height" corresponding to its fitness value. A trajectory of the system through the state space will move from a given state s to that neighbouring state for which F is optimal[3]. The fitness function represents the degree to which a certain state is "preferable" to another state and transforms the state space into a so-called *fitness landscape* with "peaks" and "valleys" (see Fig.2.1.). The attractors correspond to the local minima of the fitness function (valleys), or to the maxima of the fitness function (peaks). The local maxima are the points of separation of the basins (valleys) that lie in between the peaks. In the Fig. 2.1, a sketch of a fitness landscape is presented. The arrows denote the directions in which the system will move and indicate the preferred flow of a population on the landscape. Points A and C are local optima, whereas point B is a global

[3] It refers to either minimal or maximal.

optimum. The red ball indicates moving from a very low fitness value to the top of a peak.

3) The type of the algorithm determines a trade-off between exploration and exploitation of a state space. The exploration refers to the search of unknown regions and the exploitation is (re)-using the previous knowledge in order to find better point as a candidate solution. The balance between these two strategies is controlled by some specific parameter of the algorithm and depends on the particular algorithm. This balance is of a great significance because the prevalence of one of the categories means the deviation of the used algorithm: an excessive exploration leads to the deviation of the original algorithm to a purely random search, whereas an excessive exploitation leads to a purely hill-climbing method.

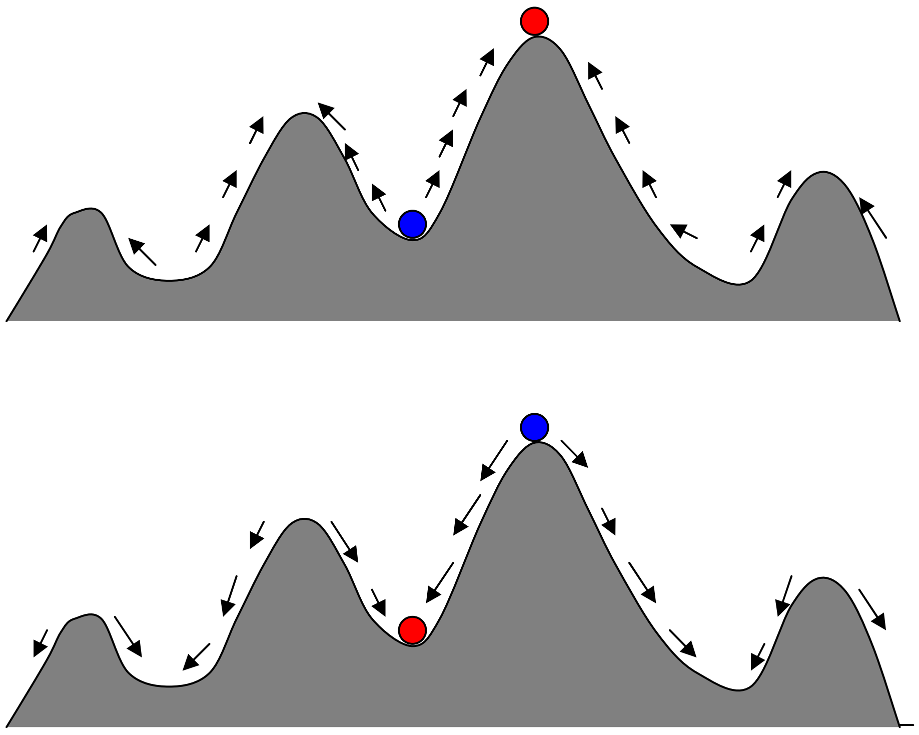


Figure 2.1: Sketch of a fitness landscape. The arrows indicate the preferred flow of a population on the landscape: (a) peaks are local optima (in case of the maximization problem), (b) valleys are local optima (in case of the minimization problem).

2.3. Summary

This chapter reviewed the state-of-the-art from the area of self-organization. Namely, self-organization has a theoretical basement as a scientific discipline, made mainly by cyberneticists, physicists and generally, by researchers from the area of complex systems. Through this chapter, it has been seen that both nature and society are prolific with different self-* mechanisms that can find the applications in IT areas. Self-organization has found the application mainly in five sub areas of IT: middleware, information systems, security, robotics and network management. This chapter emphasizes a retrospective to self-organization in P2P systems and space based computing. Especially, P2P unstructured systems are suitable as they support dynamic processes. Space-based middleware uses a blackboard based communication for the interaction of autonomous peers. Spaces have proven to be useful for communication between autonomous agents ([KMKS09]). Space-based computing offers a highly agile software architecture style ([MoKS10]). “Agile” means flexibility and robustness against unavoidable changes of requirements or changes in the environment, because the coordination is clearly separated from the business logic. Although promising approach, self-organization is without enough IT applications yet. Many mechanisms that range from nature ones to socio and business ones can be mathematically modelled and implemented. In modelling and mapping, the most suitable types of algorithms are metaheuristics.

CHAPTER 3

3 APPLICATION SCENARIOS

Although there is no doubt that complexity in IT systems exists and could be treated by using self-* approaches, it is not applicable to each scenario. The ways of discovering which problem is possibly suitable for the self-organizing application is discussed. In this chapter, two characteristic scenarios are presented. First scenario is placement, location and information retrieval in the Internet. Second one is dynamic load balancing in heterogeneous systems. Further, it is described what kind of complexity exists in these scenarios and how complexity could be measured.

3.1 Applicability of self- approach*

Self-* approach is an attractive and promising relatively new area of research, but not all IT problems can benefit of it. As this approach is proven to cope with complexity, there is no need to apply it on those IT problems that could be solvable on a simpler and traditional way. The main question is how to determine whether or not to apply principles of self-organization on a particular case, i.e., how to define the group of IT problems that could benefit of self-* principles and how to know that self-* principles will really have an effect on a particular problem. First, it is necessary to discern what kind of complexity exists in a particular IT problem (more about different types of complexity is presented in 3.4). According to that information, the conclusion can be made about what self-* mechanisms or principle could be suitable for a particular case. For example, if a considered problem possesses programming complexity (and additionally a system itself is rather heterogeneous, like a distributed heterogeneous system), then a high level of autonomy and decoupling is necessary to show some success in coping with this complexity.

The problems considered in this thesis are: dynamic load balancing in heterogeneous systems and location and information retrieval in the Internet. Both scenarios contain NP-hard problems of search and optimization.

In dynamic load balancing in heterogeneous systems, a system is heterogeneous and the appropriate load balancing algorithm considers an NP-hard problem in combinatorial optimization that includes both searching and optimization. In location and information retrieval in the Internet, the situation is the same. Therefore, self-* approach fit for both IT scenarios.

3.2 Location and Retrieval of Information in the Internet

Location and retrieval of information in the Internet nowadays becomes a more and more complex and difficult task that faces many challenges and copes with a highly dynamic nature of the Internet. An additional challenge is the manipulation of complex data (their efficient storing, querying and processing) imposed by an increasing complexity of systems and real-life applications. This requires an advanced approach that is able to manage and solve the above-mentioned problems in an autonomous, intelligent manner and that is sufficiently adaptable.

Some P2P techniques like ([ACDDHSP03], [BaBK02]) use a hierarchical addressing similar to DNS mechanism, but unfortunately the implied costs are high. ([KMGBT09]) addresses the need for efficient storage of complex structured data by proposing an architecture which unifies the P2P approach, particularly Distributed Hash Tables (DHT), with the space based computing paradigm. In ([KMGBT09]), if a retrieval of *k* entries is needed, one single DHT lookup is necessary to locate a *container* ([KMKS09]) holding structured data on which one query is performed, instead of *k* lookup queries in a plain DHT storage approach - as distributed hash tables cannot support a complex query, (i.e., limited query capability). In ([ApBu05],[BCDDD05]), the searching mechanisms in P2P overlay networks are described. In ([BCDDD05]), the algorithm has been inspired by the simple mechanism of the humoral immune system and applied to an unstructured overlay network, whereas in ([ApBu05]), the focus is put on the structured overlay network (CAN) enhanced by a mechanism that borrows some ideas from ant colony. However, generally, the issue of finding several data efficiently and concurrently can be improved, especially in a highly dynamic environment.

The proposed solution takes advantage of unstructured P2P networks, space based computing and swarm intelligence. The link between these three technologies (by means of their combination) and the proposed solution is as follows: the searching and retrieving in our unstructured P2P overlay network is realized by using swarm intelligence, whereas space based computing is used for the implementation of (sub)spaces, so-called containers, in the overlay network (to store the content that is searched for). Swarm intelligence could help highly dynamic systems to cope with environmental changes. It provides some properties inherited from biological systems: every item in the population makes local decisions, and behaves and acts in a decentralized manner. The subjects in this architecture are software agents that, e.g., perform the roles of artificial ants. The architecture is a new overlay network with an intelligent lookup mechanism based on swarm intelligence that is able to cope with complex queries even if the “full information” is not available in the given query.

3.3 Load Balancing in Distributed Heterogeneous Systems

Load Balancing can be described as finding the best possible workload (re)distribution and addresses ways to transfer excessive load from busy (overloaded) nodes to idle (under-loaded) nodes. The goal is to distribute workload evenly across a network in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid overload. It is possible to talk about load balancing at local node level allocating load to several core processors of one computer, and load balancing at network level distributing the load among different nodes. The second case is more complex and requires an additional estimating of the priorities as the transfer of data itself from a busy node to an idle node could be more time consuming than the load assignment.

Many different approaches cope with the load balancing problem. One classification can be done according to the used load balancing algorithm and the approaches can be classified in conventional, pure theoretical and advanced (intelligent) ones.

The first group consists of different conventional approaches without using any kind of intelligence, e.g.: Sender Initiated Negotiation and Receiver Initiated Negotiation ([ShKr94]), Gradient Model ([LiKe87]), Random Algorithm ([Zhou88), and Diffusion Algorithm ([CRCSL02]).

Sender algorithm has a good performance for low to moderate load levels. In this algorithm, an overloaded node is responsible for initiation of load balancing. Just opposite, Receiver algorithm has a good performance for moderate to heavy loads levels - load balancing is initiated by the under-loaded node. Symmetric algorithm refers to the combination of these two algorithms in order to "conciliate" these two opposite situation, to make the bridge between them and overpass disadvantages from both sides.

In Gradient Model, an underloaded node dynamically initiates load balancing requests, which result is a system wide gradient surface. Overloaded nodes respond to requests by migrating unevaluated tasks down the gradient surface towards underloaded nodes.

In Random Algorithm, load balancing is basically initiated by an overloaded node and the "partner node" is chosen randomly without taking in consideration whether the target "partner" node is overloaded or not. Namely, each node checks the local workload during a fixed time period. Once when a node becomes overloaded, it sends the newly arrived task to a randomly chosen node.

The principle of diffusion algorithms is keeping the process iterate until the load difference between any two processors is smaller than a specified value.

The second group includes pure theoretical improvements of load balancing algorithms by means of developing the advanced mathematical models ([BrMe008]). The disadvantage is a lack of practical proof-of-concepts, implementations and benchmarks.

The third group contains advanced approaches that use intelligent algorithms like evolutionary approaches ([ChLH08]), and ant colony optimization approaches ([HoEw07]). Evolutionary approaches rely on their high level of adaptation and use the adjustment of specific parameters in order to achieve the goal of load balancing. Ant colony optimization approach is used in for a graph theoretic problem ([HoEw07]), i.e., Ant Colony Algorithm is applied to the problem of constructing load-balanced clusters in ad hoc networks with node mobility. The intelligent algorithms from the last group showed promising results.

Nevertheless, they still need improvement concerning experience in tuning of algorithms, quality of solution they provide, scalability, provisioning of a general model, flexibility, etc. ([LDTN08]) compared non-pheromone-based (bee intelligence) versus pheromone-based algorithms. Its conclusion is that the former are significantly more efficient in finding and collecting food.

3.4 Complexity in application scenarios

The *sources* of complexity in the presented application scenarios are:

1) amount of resources, i.e., the huge amounts of distributed components that must interplay in a global solution,
2) type of resources, i.e., heterogeneity
3) large number of interactions of the various elements of the software
4) huge problem size (like number of computers-which correlates to 1), clients, requests, size of queries etc.)
5) autonomy of organizations
6) dynamic changes of the environment (complex adaptive systems).

According to these sources, different *types* of complexity can be discerned: point 1) is a pure *computational complexity*, points 2) and 3) address *programming complexity*, point 4) refers to both computational and programming complexity, and 5) and 6) are the consequences of features of complex adaptive system.

In the analysis of computational complexity ([FoHo03]), two well-known types appear:

- *time complexity* - the length of time it takes to find a solution or complete a process as a function of the size of the input;

- *space complexity* - the amount of physical storage required for a system to perform a certain operation, i.e., to solve an instance of the problem as a function of the size of the input.

Every task[4] can contain subtasks. When all subtasks are carried out in a required order and completed, consequently the task is successfully completed. The order of complexity of the task is determined through analyzing the demands of each task by breaking it down into its constituent parts. All tasks should fit in some configured sequence of tasks, making it possible to precisely determine the hierarchical order of task complexity. It is based on a complex mathematical model of how the information is organized ([CoGD97]). Tasks vary in complexity in two ways: either as *horizontal* (involving classical information) or as *vertical, i.e., hierarchical* (involving hierarchical information). Horizontal complexity is the amount of information in simple quantitative terms within a task. It consists of the number of different responses that have to be performed. For example, if the number of bits for a representation of some number is considered, then counting to 2 is one bit, 4 is 2 bits, 8 is three bits, 16 is 4 bits. Hierarchical complexity refers to the number of recursions that the coordinating actions must perform on a set of primary elements. The actions at a higher order of hierarchical complexity: (a) are defined in terms of actions at the next lower order of hierarchical complexity; (b) organize and transform the lower-order actions; (c) produce organizations of lower-order actions that are qualitatively new and not arbitrary, and cannot be accomplished by those lower-order actions alone. Once these conditions have been met, we can say the higher-order action coordinates the actions of the next lower order. For example, consider the action A_1 of evaluating a + b and the action A_2 of evaluating (a + b) + c. The horizontal complexity of A_1 is smaller than the horizontal of A_2 since the action of addition is executed less often in A_1 than in A_2. On the other hand, because A_1 differs from A_2 only in how many times addition is executed, but not in the organization of the addition, both actions have the same hierarchical complexity.

So, in the presented application scenarios, the above mentioned types of complexity can be observed (programming and computational, in which both time and space complexity are present; additionally hierarchical complexity is present). Both scenarios consider the problems of searching in the network and the optimizing a path at the same time.

Generally, problems can be classified by complexity class according to the time it takes for an algorithm to solve them as a function of the problem size. Some problems are difficult to solve, while others are easy. The problems that are described in these scenarios are NP problems, i.e., difficult problems that need algorithms that take an exponential amount of time in terms of the size

[4] The notion of a task is used here as an example, and could be generalized with the notion of a process.

of the problem to solve. For sure, it depends on the size of the network. As the size of the network grows, the time needed to find the route grows (more than) exponentially. Even though a problem may be computationally solvable in principle, in actual practice it may not be that simple. These problems might require large amounts of time or an inordinate amount of space.

3.5 Measurement of Complexity

The title refers to both measurement of complexity and measurement of self-organization as there is a strong correlation between these notions.

Researchers from different areas of science like biology, computer science, finance, etc., define different measures of complexity for each respective field. ([Lloy01]) presents a categorization of complexity measures by defining common questions for all problems:

1) How hard is to describe?
2) How hard is it to create?
3) What is its degree of organization?

([Lloy01]) provides a list of some measures of complexity grouped according to the question that they try to answer. In case of difficulties of descriptions, some kind of entropy could be used as a metric. If a difficulty of creation contributes to complexity, then it is typically measure in time, and it usually addresses computational complexity (time computational complexity and space computational complexity). A degree of organization can be divided into two quantities:

a) Difficulty of describing organizational structure,

b) Amount of information shared between the parts of a system as the result of this organizational structure.

In case of difficulty of describing organizational structure, some kind of metric entropy is used. Specific types of complexity (hierarchical, stochastic, homogeneous) are connected with this issue. Part b) addresses mutual information and it is strongly connected with measurements of algorithmic mutual information, correlation and organization itself. [5]

The concept of measurement of self-organization is strongly related to the concept of measurement of complexity. From the theoretical point of view, the ratio of the volume of the basin to the volume of the attractor can be used as a measure of the degree of self-organisation present. This Self-Organization Factor (SOF) will vary from the total size of state space (for to-

[5] The above mentioned measurements and metric refer only to the IT case. Generally, various types of complexity measurements can be found that address 1), 2) and 3) for different scientific disciplines.

tally ordered systems - maximum compression) to 1 (for ergodic - zero compression).

([Heyl01]) formally describe uncertainty and entropy in the following way:

All states have the same probability: $P(s) = P(s')$, $\forall\ s, s' \in S$. This implies that if the system is in a particular state s: $P(s) = 1$, then $P(s') = 0$, $\forall\ s' \neq s \in S$. Shannon[6] (information) entropy can be used to determine a degree of uncertainty H about the system:

$$H(P) = -\sum_{s \in S} P(s) \log P(s) \qquad (3.1)$$

Note: the sum can be replaced by an integral in case that the state space is continuous. Uncertainty could take minimal, maximal and intermediate values in these situations:

- minimal: $H = 0$, when one state has probability 1 and all others have probability 0,
- maximal: $H = \log N$, (where N is the number of states in S), when all states have the same probability,
- an intermediate value: $H = \log N_0$, with $1 < N_0 < N$ the number of states in S_0, when the state reside within a subspace $S_0 \subset S$.

Self-organization is equivalent to the reduction of H, that can be calculated as H(before) - H(after).

([VaBr01]) investigate emergent self-organization in multi-agent systems trying to connect with the second law of thermodynamics. It appears to be contradictory as the macro level that hosts self-organization has an apparent reduction in entropy, whereas in the micro level, where random processes exist, greatly increase entropy. They define a way to measure the Shannon entropy at the macro (agent) and micro (pheromone) levels. The example of different levels in one self-organizing system is an ant colony. Ants and their movements constitute the macro level of the system, while pheromone molecules constitute the micro level. The construction of minimal paths between their nests and food sources achieve a reduction in disorder at the macro level. This is possible because the agents at this level are coupled to the micro level, where the evaporation of pheromone molecules results in a growth in disorder. As a result, the disorder of the overall system increases. Through the notion of thermodynamic entropy, they introduce information (Shannon) entropy (Eq.3.1.) stating that the first one has strong formal similarities to the second one. Further, computing the Shannon or information entropy (defined in Eq.3.1.) requires measuring the set of states accessible to the system and the probability of finding the system in each of those states. ([VaBr01]) define

[6] Shannon's uncertainty and Boltzmann's statistical entropy are equivalent.

state, and thus entropy, in terms either of location or direction: location-based state is based on a single snapshot of the system, while direction-based state is based on how the system has changed between successive snapshots.

([ShSh03]) find that self-organization is the same as a spontaneous increase in complexity, leaving us with the problem of measuring complexity. The obvious candidate, from a physical point of view, is thermodynamic entropy, but unfortunately it seems to be a most unsatisfying measure of organization in complex systems: It is quite difficult (and seems to be impossible) to establish unique entropy for all self-organized systems. Thermodynamic entropy measures how far a system departs from being in a pure state [ShSh03]. From the other side, different examples of organization in different science disciplines cannot fit to this definition of entropy. For example, in the systems treated by statistical mechanics, pure states are more organized than impure ones, whereas in biology, the systems of organisms are never in pure states.

Grassberger ([Grass86]) proposed the idea that defines the complexity of a process as the minimal amount of information about its state needed for maximally accurate prediction. The Crutchfield-Young "statistical complexity", $C\mu$, of a dynamical process is the Shannon entropy (information content) of the minimal sufficient statistic for predicting the process's future. For a full exposition of the resulting theory, as it applies to classical stochastic processes, see [Shal01].

Obviously, a general form of self-organization measurement doesn't exist. For example, in ([CMVT07]), the mechanism of "brood sorting" is used and spatial entropy is proposed as a measure of self-organization.

In this thesis, the measurement of self-organization, i.e., how good the single contributors (bees, ants, …) organize themselves is realized by means of the suitability function, i.e., similarity function. The description of these functions is in Chapter 5, connected to the description of the algorithms. Higher values of these functions denote the better self-organization in the presented systems. Computational complexity is tracked in time.

3.6 Summary

The location of problem types where self-* can be useful is the starting important step. Chapter 3 described the ways of detecting use cases suitable for the application of self-* mechanisms. First, it should be clear which type(s) of complexity exist in a particular IT problem. According to that, it is possible to conclude what self-* mechanisms would be the best applicable (they range from autonomy and decoupling like mechanisms in P2P systems and space based computing to intelligent algorithms). Also, problems that are NP-hard and require an optimization need to be treated by self-* approaches, especially if large instances are taken in consideration. In this chapter, two use cases are detected and considered in this thesis: dynamic load balancing in heterogeneous systems and location and information retrieval in the Internet. Both scenarios contain NP-hard problems of search and optimization, and therefore they are applicable to the self-* approach. After defining types of complexity in these scenarios and analyzing the measurements of complexity, it is concluded that a general measurement of complexity (and consequently, of self-organization) cannot be derived. It is problem and domain specific. In this thesis, the measurement of self-organization is based on functions specially constructed for each considered case.

CHAPTER 4

4 DESIGN AND IMPLEMENTATION

The necessary frameworks for the application scenarios are presented in this chapter. The first one, P2P Unstructured Intelligent Overlay is a framework for location and information retrieval in the Internet. The second one, SILBA (self initiative load balancing agents) is a more complex framework that serves for dynamic load balancing. After explaining their design and architectures and identifying patterns, the communication in the architectures is described by using the PlusCal algorithm language ([Lamp09]) and further, models are checked via TLC model-checker ([Lamp09]). At the end, the way of their implementations is explained by using sequence diagrams. Further details about the implementation can be found in the Appendix A. The complete PlusCal algorithms constructed for both architectures and their TLA+ translations are presented in Appendix B.

4.1 Architectures

The framework for information retrieval in the Internet represents a purely distributed overlay network based on spaces (XVSM containers) and can be viewed as a pattern for plugging different swarm algorithms. In this work, ant algorithms and bee algorithm are plugged.

4.1.1 P2P Unstructured Intelligent Overlay

The **architecture and design**, used for the application scenario - Information Retrieval from the Internet, is described in the following way ([ŠeKü09], [ŠeKü10a]):

When a container is created, it becomes accessible to the public by its URL - it can be located and accessed via this URL and all operations (*read, take, write*) refer to the container by its URL. In this way, no explicit connection to a space is needed. The improvement of this access of containers by introducing an intelligent lookup mechanism is investigated.

The pre-assumptions are:

1) No overlapping containers exist.

2) An entire container must belong to one node, i.e., one node can contain one or more complete containers; but it is not possible that one part of the container belongs to one node and another part to some other node.

3) Container C can contain "subcontainers" in a sense that the entries of C contain references (URLs) to some other containers.

The first step in designing an intelligent lookup is to create an overlay network that consists of lookup containers.

The coordination space serves to implement lookup containers. A container is published under one or more *published names*. Therefore, the container is reachable by its URL and additionally by its published names. Each entry stores a published name and a URL of some container, and possibly some additional information of its properties.

Example: An entry without additional information can be
[published-name="test", URL="www.test.ac.at"].

The lookup is performed by means of swarm intelligence. When some specific container is needed to be found, then the searching is done through lookup containers according to the published name and the result is the URL retrieved from one of the given lookup containers[7].

Structure of the Overlay Network. The chosen overlay is a purely decentralized and unstructured one. For an initial construction, the scale-free network approach ([Cald07]) as a common, real wide-spread approach is used with the initial number of containers, $m_0 = 2$, according to the Barabási–Albert model that is an algorithm for generating random scale-free networks ([AlBa02]). The network is dynamically created. URLs are retrieved by using published names. The relationship between URLs and published names is 1: *n* relation.

A node is a computing device that might consist of several core processors. An autonomous agent is a software program that is self-responsible to be up and running. An agent implements a certain reactive and continuous behaviour ([DDFG06]). Agents can move from node to node, and they can dynamically join and leave. A client issues requests at any reachable node in the network. A design pattern is a reusable solution in software design ([HoWo03]).

The main architecture is represented in Fig 4.1. (ovals represent space containers, rectangles represent software agents). This figure represents the local node pattern (the so-called swarm node). The whole network consists of finite numbers of swarm nodes. As both ant algorithms and bee algorithm are implemented, the swarm node is the ant node (and the architecture refers to the ant space, the ant agents and the ant answer space) in case of the ant algorithms, i.e., the swarm node is the bee node (and the architecture refers to the bee space, the bee agents and the bee answer space) in case of the bee algorithm. The local node models the content which is the subject of search and provides an environment for its searching by using swarm agents (ants or

[7] The manipulation of data in the overlay network uses the XVSM core API and query language.

bees). The components of a local node are: clients, swarm agents, a swarm space, a content space and a routing space.

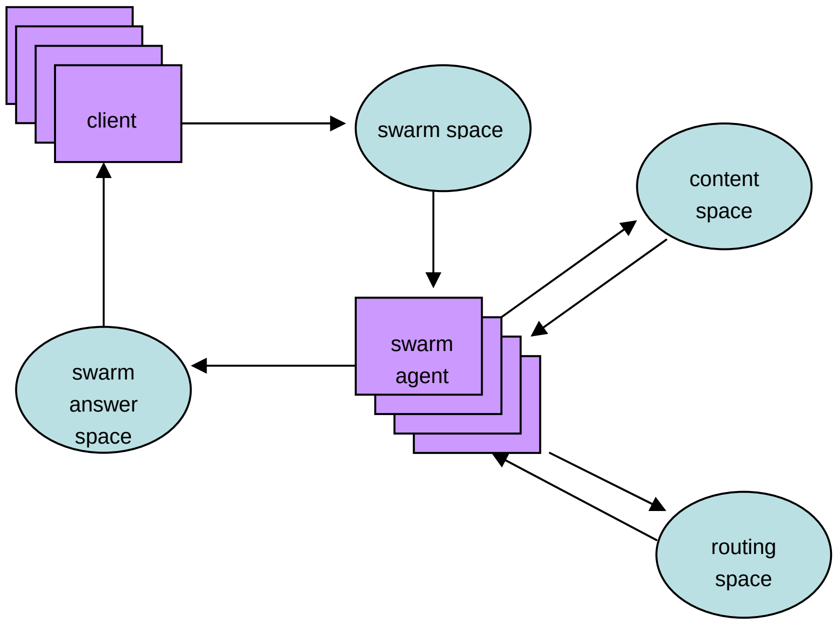


Figure 4.1. Local Node Pattern

Clients supply the requests. The swarm space is dedicated for putting the searching requests and contains the information about the current status of searching and the current list of visited nodes. The swarm agent receives the request, realizes the search and changes the status of the search according to the quality of data contained on a particular node (no_data, acceptable_data, exact_data) by using data quality policy which is implemented by the similarity function δ (explained in detail in 5.2.3.) and presented in Fig. 4.2. Swarm agents consult a content space and a routing space. The content space contains the information about the public name(s) and the real name(s). The routing space contains the list of neighbouring nodes (like a routing table) and the additional information connected with the type of algorithm (e.g., quantity of pheromones for ant algorithms, duration of waggle dance for bee algorithm on links). At the end, the result is put in the answer space, where it is picked up by clients.

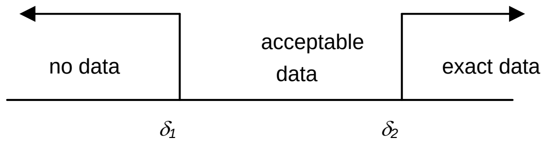


Figure 4.2. Node classification according to the data quality policy.

The next figure 4.3. presents the pattern composition in case of ant algorithms. Namely, in case of ant algorithms, there are two types of behaviour of an ant agent: forward and backward. The elements of swarm (ant) pattern are swarm (ant) agents and a data quality policy expressed through the similarity function. This policy estimates the quality of information on the current node. As denoted, there are two types of ant agents:

- FORWARD – the agent searches the local content; in case that the desired content is not found, it goes to the next (neighbouring) node by using the information from the routing space.
- BACKWARD – if the desired content is found, the agent gets status "backward" and takes the same way back (i.e., the same path that forward agent used, bit in opposite direction).

The same figure can be used in case of bee algorithm. The difference is that a swarm (bee) agent has no explicitly separated state of forward and backward. The whole trip forth and back is encompassed in the so-called navigation phase of an agent (chapter 5). So, the violet rectangle in Fig.4.3. (without forward and backward designation) can represent a swarm bee agent.

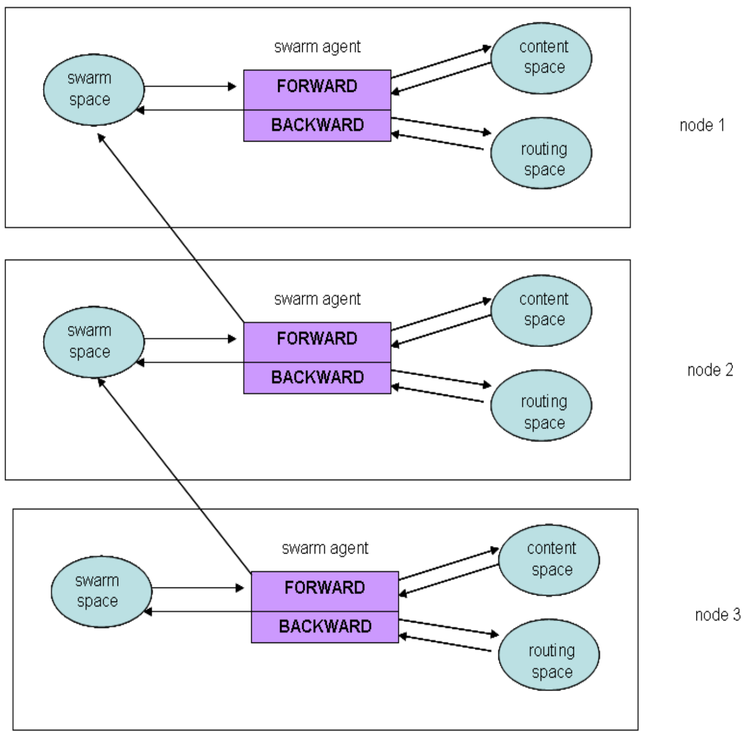


Figure 4.3. Pattern composition (for ant algorithms).

4.1.2 SILBA

The approaches, numbered in 3.3., mainly try to improve only one of the components of the whole load balancing infrastructure, namely the load balancing algorithm. The original problem includes missing of a general framework that is the implementation of a complex pattern composed of several sub-patterns that represents a general, re-usable architectural solution to a certain problem scenario. A framework can be measured by its architecture agility ([KüŠe09]).

A comprehensive classification of different load balancing approaches can be found in ([KüŠe09]), where it is referred to the problem as a lack of provisioning a general framework, autonomy, self-* properties, and arbitrary configurations.

Some interesting approaches are based on autonomous agents and multi agent technology. According to multi-agent systems, load balancing can be divided into these categories ([GoSc01]):

- A static load balancing: once the tasks have been launched on a specific server, they cannot be migrated elsewhere.
- A mobile load balancing: a task may migrate to another server, utilizing the agent's mobility.

Table 4.1. Classification of load balancing approaches ([KüŠe09])

	framework abstraction	no framework
agent based	[HCCD05]*, [WaLi03], [TLYD05], [RaHa00]	[GeRa03], [TSTNT05]*, [JoDK02], [HeBP07]
without agent	[BCCP04], [BaCh03]	[GLSKS04], [KaRu04], [MTYK06], [Putr03], [Rahm08], [XuGu07], [XuBh06], [ZhHu07], [ZoDK07]

The above table gives another type of classification through two “filters”, i.e., according to two criterions: whether an approach uses agents or not, and whether an approach possesses a general framework or not.

([DaJS06]) presents fundamentals of different multi-agent architectural styles, shows how they can be characterized and evaluated, and considers whether an approach introduces a general framework (taking in the consideration both structured P2P and unstructured P2P networks as well as grid). Those papers that use a very specific load balancing algorithm are marked with “*”. For example, ([Rahm08]) puts the focus on DHT-based P2P systems only.

Each of categories, presented in table 3.1, is discussed:

- *Agents and framework*: ([HCCD05]) uses a very problem-specific load balancing algorithm with a focus on parallel database systems. ([WaLi03]) presents a dynamic behaviour of agent-based load balancing on grids as well as modelling and predicting load balancing behaviour in order to explore the effects of agents' strategies on the quality of load balancing. A disadvantage is that the model used for abstraction is not very generic. ([TLYD05]) is also problem-specific, in particular designed for parallel database systems, and not generic. ([RaHa00]) provides a generic approach to implement any kind of dynamic load balancing algorithm in a heterogeneous cluster using software agents. A disadvantage in this approach is that it strictly uses only sender-initiated algorithms (no generalization is shown that allows plugging in also other algorithms).
- *Agents, no framework*: ([HeBP07]) describes the AMBLE model, an awareness model which manages load balancing by means of a multi-agent based architecture, with the aim to establish a cooperative load balancing model for collaborative grid environments, and presents its extension, named C-AMBLE (Cooperative Awareness Model for Balancing the Load in grid Environments). It applies some theoretical principles of multi-agents systems, awareness models, and third party models, to promote an efficient autonomic cooperative task delivery in grid environments.
- *No agents, framework:* ([BCCP04]) describes the design of a flexible load balancing framework, named PREMA, and runtime software system for supporting the development of adaptive applications on distributed memory parallel computers. An indication of the flexibility of the PREMA system has been given by implementing several load balancing policies.
- *No agents, no framework*: ([Putr03]) states that load balancing at the middleware level allows more flexibility than existing solutions based at lower system levels. DLBS (Dynamic Load Balancing Service) consists of a scalable monitoring infrastructure, a connection manager (integrated into the middleware) and customizable load balancing strategies. It brings new solutions regarding large scale load balancing for middleware-based applications, and offers a multi-criteria and easily customizable load balancing service.

Hence, the list of issues that an advanced approach needs to comprise and provide is the following issues:

- *An intelligent load balancing algorithm*: Existing load balancing algorithms are not powerful enough to cope with high dynamics and complexity in nowadays systems. New intelligent approaches are needed to support adaptation and improve performance and scalability.

- *A General Framework:* Existing load balancing approaches are very problem- specifically oriented and therefore, their comparison is very difficult or even impossible to be done. A new approach needs to be "omnipotent" and undependable of the specific problem. Therefore, in order to find a best solution for a problem, a generalized framework is needed that allows for testing and tuning different load balancing algorithms for a specific problem and environment.
- *Autonomy and Self-* Properties:* Increased complexity of software systems, diversity of requirements, and dynamically changing configurations, force to find new solutions based on self-organization, autonomic computing and autonomous (mobile) agents. Intelligent algorithms require autonomous agents which are advantageous in situations that are characterized by high dynamics, not-foreseeable events, and heterogeneity.
- *Arbitrary Configurations*: As it is already stated, load balancing can take place at a local level, to manage the load among local core processors on one node, as well as at a network level (intranet, internet, cloud). A general load balancing framework must support arbitrary configurations and be able to cope with all these demands at the same time by offering means to abstract hardware and network heterogeneities.

Load Balancing functions through different policies. Two of them are the main ones: transfer policy and location policy. A transfer policy determines whether and in which form a resource should participate in load distribution and in that sense, the classification of resources is done ([ShKr94]). A simple transfer policy would be to define two parameter values T_1 and T_2 (Fig.4.4.) which can either be assumed to be static or can be changed dynamically. A location policy determines a suitable partner of a particular resource for load balancing ([ShKr94]).

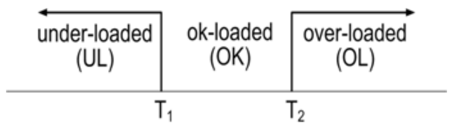


Figure 4.4. The classification of nodes according to the Transfer Policy ([KüŠe09])

This research focuses on a new conception of a self-organizing coordination infrastructure that suggests a combination of coordination spaces, self-organization, adaptive algorithms, and multi-agent technologies. Each of the numbered issues has some form of self-organization in its incentive. In the following, a load balancing framework termed SILBA (self initiative load balancing agents) is described in detail. SILBA supports dynamic exchange of algorithms and combinations of different algorithms (both unintelligent and intelligent[8]). The underlying logic is to serve as a test bed to ease the selection of the best algorithm for a certain problem scenario under certain conditions. SILBA supports load balancing on several levels (e.g., between nodes in network, between subnets in a network as well as between different networks) and allows for combinations of different algorithms on different levels (e.g., swarm intelligence algorithms on each level or combination of swarm intelligent algorithms with unintelligent algorithms). The architecture is agile, i.e., new requirements on load balancing algorithms, the network infrastructure, dynamic processes (like joining and leaving of agents) do not influence to the stability of the architecture and do not become “architecture breakers” ([MoKS10]).

Basic SILBA Design

The SILBA framework ([KüŠe09]) allows the exchange of pluggable algorithms and supports their combinations. The SILBA itself does not solve the load balancing problem - it serves as a “platform” to ease the selection of the best algorithm for a certain problem scenario. It is based on decentralized control and a blackboard based style of collaboration, which is the starting point for the realization of self-organization. The SILBA pattern is domain independent and can be used on different levels.
<u>A local node level</u>. A load is allocated to several core processors of one computer. In this case, load balancing refers to distribution of load between all core processors and their balanced utilization.
<u>A network level</u>. It refers to the distribution of the load among different nodes and can be extended up to several sublevels: load balancing between different subnets inside one network and load balancing between different networks.

The basic components of the SILBA model are clients, autonomous agents, tasks and policies. *Clients* request for tasks to be executed and they are responsible for the load, present in a network. *Autonomous agents* operate in a P2P network, dynamically exchanging amount of work between nodes. In this model, different types of autonomous agents exist (they are described in the further text). A *task* can be described as a tuple:

(*priority*, *job*, *description*, *properties*, *timeout*, *answer space*), where

- the priority pinpoints to the importance of a task,

[8] By using the notion of “intelligent algorithms” it is often referred to the class of algorithms from artificial intelligence based on some kind of swarm intelligence or evolutionary computation.

- the job is expressed in a XML- or WSDL-format,
- a semantic description is optional parameter,
- properties are specific for a certain task and could be, e.g., whether task's execution mode is "at-most-once" or "best-effort",
- a timeout, and
- an answer space is an URL of an Internet addressable resource where to write the result of the execution back.

Two main policies, transfer policy and location policy, are defined in chapter 3.

The SILBA pattern is composed of the following sub-patterns (Fig.4.5).

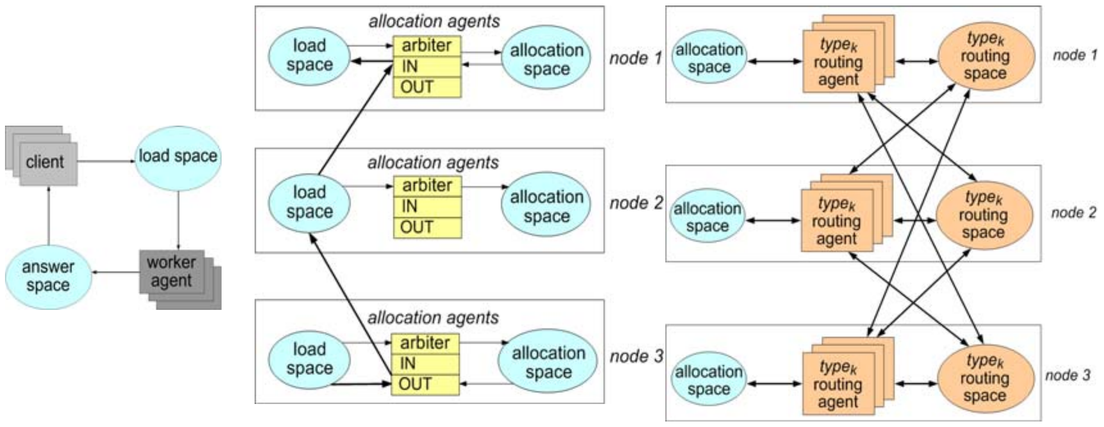


Figure 4.5. Patterns in SILBA: (a) local node pattern (b) allocation pattern (c) routing pattern ([KüŠe09])

The execution of requests by local worker agents takes place in the **local node pattern** (Fig4.5(a)). The basic components of the local node pattern are: clients, worker agents, load space, and answer space. The requests, put by client(s), are accessible in either the order they arrived, or by means of other criteria, (e.g. their priority, the required worker role, or their timeout date). Worker agents actively and constantly compete for a work. In a load space, new requests are put by clients, and the information about all worker agents' registrations and the current load status (UL, OK, OL) of a node are maintained. So, this is the place where transfer policy is continuously updated. In Fig 4.4., a very simple transfer policy is presented. It is extended by the following parameters: individual criteria (T_0); a time threshold parameter (T_1) - if time exceeds this threshold, the task is rescheduled according to the location policy; threshold parameters (T_2 and T_3) marking how many tasks one worker agent may have in order to be under-loaded (UL), ok-loaded, (OK), or overloaded (OL). The transfer policy is executed by each worker agent autonomously based on some individual criteria (T_0). Finally, the answers that were computed by the worker agents are put directly in the answer space, so that they can be picked up by the clients.

Redirecting a load between the load spaces of different local nodes is governed by the **allocation pattern** (Fig4.5(b)). The basic components of the allocation pattern are: load space, allocation agents, policies, and allocation space. There are three types of allocation agents: arbiter agents, IN agents, and OUT agents. Arbiter agents query the load of the load space and publish this information to the routing space. Both IN and OUT agents read routing information from the allocation space and pull/push work from/to another node in a network to which the current node has a connection. The allocation space holds information about the best partner nodes as computed by the location policy.

The **routing pattern** (Fig4.5(c)) is responsible for the execution of the location policy according to a particular load balancing algorithm. The basic components of the routing pattern are: allocation space, routing agents, and routing space. In a more complex situation where the combination of algorithms takes place, routing agents can be of different types. They mutually communicate and collaborate with other routing agents, but only of the same type, via the corresponding routing spaces of this type. Each type of routing agents has its own routing space where specific information, required by the applied algorithm, is stored and retrieved[9]. The information about the best or suitable partner node is stored in an allocation space, where the corresponding IN or OUT allocation agents can take this information and distribute the load between the local node and its partner node.

The full complexity of this architecture can be viewed in a **pattern composition**. Namely, all above described patterns can be composed by connecting them via shared spaces. The agreement about format of entries stored in these spaces is the prerequisite.

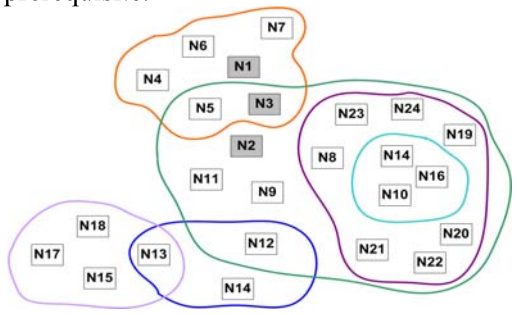


Figure 4.6. An example of a network configuration ([KüŠe09])

[9] For example, this specific information can be pheromones, if ant algorithms are used as LB algorithm, or duration of waggle dance, if bee algorithm is used as LB algorithm.

As an example, Fig.4.6. represents seven networks that have different relationships to each other. The nested networks and the intersection between two (or more) networks are allowed. Therefore, nodes can belong to one or more networks, e.g., nodes N1 and N2 are part of one network each, whereas N3 belongs to two networks.

Extended SILBA Design

The design of the basic SILBA can be extended towards remote load balancing. Although the patterns for different levels of load balancing are the same at each level, they are parameterized by other algorithms. The extensions of the SILBA framework allow for load balancing on several levels.

Example: The extended SILBA with two levels can be described as follows:

- SILBA level$_1$: between different subnets, simultaneously with load balancing between nodes within these subnets, and
- SILBA level$_2$: between different networks, simultaneously with load balancing between subnets in these networks and nodes within these subnets.

Different algorithms (hybrid algorithms, their combinations) can be applied on each level and load balancing in the whole network can be realized through the combination of algorithms. Fig.4.7. represents one network topology configuration example. The overlapping of different subnets as well as the existence of nested subnets is allowed and supported. In the following text, two levels of load balancing are described. Note that the SILBA is not limited with the number of levels and can be further extended to n levels ($n \geq 1$, $n \in N$).

The extended SILBA introduces more benefits:

1. More efficient load balancing and an improving of the overall system performance;
2. A concurrent exchange of different load balancing algorithms on different levels, which leads to construction of their combinations and hybrid forms; therefore, it is easier to find the best combination of algorithms for a particular problem;
3. The approach is domain independent, problem independent and could be used in an arbitrary network structure.

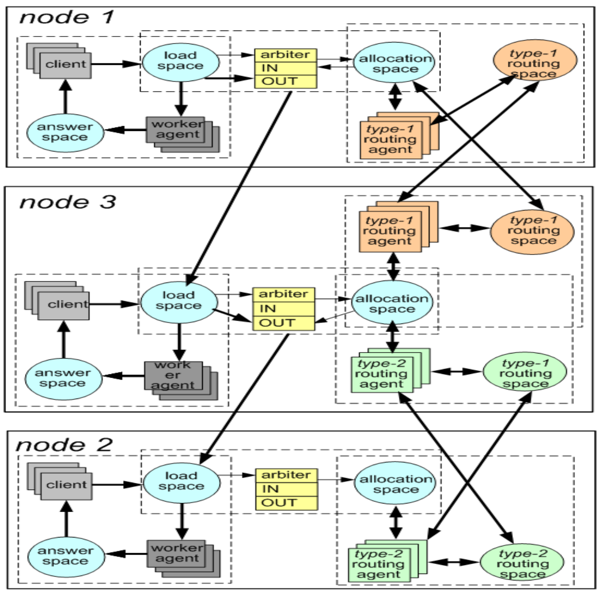


Figure 4.7. An example of a network topology ([KüŠe09])

Load balancing in a Subnet (level$_1$)

An extended behaviour of a routing agent is necessary. For example, in Fig.4.5, node N3 belongs to two different subnets. In one subnet, routing agents are of type$_1$ (i.e., they implement one load balancing algorithm) and in the other subnet routing agents are of type$_2$ (i.e., they implement another load balancing algorithm). So, in order to collaborate with nodes from both sub-nets, node N3 must posses both types of routing agents (incl. both types of their routing spaces that hold the information specific for each load balancing algorithm respectively). The collaboration between different types of routing agents at node N3 goes through its allocation space. As the allocation space holds the information about partner nodes (computed by the location policy), the IN and OUT allocation agents assume that the information about the best partner to/from which to distribute load can be queried from the allocation space.

Load balancing between Subnets (level$_2$)

The next level of load balancing includes a further extension of routing agent behaviour for the internet routing. Each routing space is published under a public name by using the publishing layer and lookup peer-to-peer (e.g., JXTA [JXTA10]) layer. A routing within a subnet uses the same pattern as routing between one or more subnets.

4.2 Model description and verification

The architectures explained in subsection 4.1. are modelled by using the PlusCal algorithm language in order to verify them. A mathematical notation for the construction by using the PlusCal algorithmic language is provided. By using TLC model-checker, the correctness of architectures is proven for all combination of algorithms that can be plugged in and all network topologies.

4.2.1 Pluscal Algorithm Language

The PlusCal algorithm language is based on TLA+ (Temporal Logic of Actions) specification language ([Lamp09]) and made with the intention to provide tools for describing algorithms, making their specifications and checking their correctness. Under the name "algorithms", a wide class is encompassed (including the algorithms that describe the architectures). So, it is possible to check the correctness of a model or architecture and verify it by using TLC model checker ([Lamp09]). TLC is a model checker for specifications written in TLA+. PlusCal can be applied both to sequential and concurrent algorithms. A translation of a PlusCal algorithm generates a TLA+ specification, and further it can be subjected to the TLC model checker.

PlusCal has the mathematical and logical basement, i.e., it is based on set theory and first-order logic. Namely, TLA+ specification is high-level specification language based on predicate logic, sets and functions. However, PlusCal is highly descriptive and expressive, and therefore can replace pseudocode. The algorithm in PlusCal can be written in two syntaxes: p-syntax and c-syntax. The algorithms presented in subsection 4.2.2. are written in c-syntax.

The algorithm written in PlusCal must be positioned in a file of a TLA+ module and it has the following structure ([Lamp09]):

```
--algorithm name
  declaration(s)_of_variables
  definitions
  macro(s)
  procedure(s)
  algorithmBody OR process(es)
  end algorithm
```

where it is possible to have 0 or 1 instance of declaration, 0 or 1 instance of definition, 0 or more instances of macro, 0 or more instances of procedure, and an algorithm body or 1 or more instances of process.

The precise grammatical rules for each of these items can be seen in ([Lamp09]).

Labels indicate atomic actions and are required in front of the first statement in the body of a procedure or a process, □□□□□ statement, in front of the statement immediately preceded by an □□ or □□□□□□ statement that contains a □□□□□ □□□□□ □□□□□□□ or labelled statement with it.

The most important PlusCal statements are ([Lamp09]):

- **Assignment** to a variable or a component
- **If** statement (the meaning is well-known):

  `if` *test* `then` *clause*$_1$ `else` *clause*$_2$ `end if`;

- **Either** statement (it refers to a nondeterministic choice, i.e., any executable clause can be chosen) :

  `either` *clause*$_1$
      `or` *clause*$_2$
      ...
      `or` *clause*$_n$
  `end either` ;

- **While** statement (the meaning is also well-known):

  `label` : `while` *test* `do` *body* `end while` ;

- **Await** statement (it can be executed only when the value of *expr* equals TRUE, otherwise it blocks):

  `await` *expr* ;

- **With** statement (it allows for a nondeterministic choice of an element from set *S*):

  `with` *el* $\in$ *S* `do` *body* `end with` ;

- **Skip** statement does nothing
- **Print** statement (TLC does printing of *expr*):

  `print` *expr* ;

- **Assert** statement (it asserts that *expr* equals TRUE and in this case it is equivalent to `skip`; in case that *expr* equals FALSE, TLC will report an error message):

  `assert` *expr* ;

  Instead of printing results (a huge number of them, as there can be many different states – often an infinite number of reachable states) and examine them by hand, assert statement is a useful and elegant tool to check the correctness of the results – TLC will do the checking automatically by using assert statement.

- **Call** statement and **return** statement are correlated with procedures
- **Goto** statement (the meaning is also well-known):

  `goto` *label*;

Further processes, procedures, macros and definitions may be parts in a PlusCal algorithm:

- **Processes**: Algorithms might be multiprocess algorithms and contain one or more concurrent processes. In case that it is required to denote a set of processes, then use

  `process` *ProcName* $\in$ *Set*

  otherwise, in case of one individual process, then use

  `process` *ProcName* = *Id*

- **Procedures:** One or more procedures might be part of an algorithm. A procedure *MyProc* will begin with

  `procedure` *MyProc* ($param_1$, …, $param_n$)

  whereas procedure's body starts with `begin` and ends with `end procedure`, and must begin with a labelled statement. A procedure is called by using a `call` statement, e.g., the previous procedure could be called as

  `call` *MyProc* ($expr_1$,…,$expr_n$) ;

- **Macros** are similar to procedures, except the fact that "a call of macro is expanded at translation time"
- **Definitions** are realized by using a `define` statement

PlusCal uses the expressions and definitions inherited by TLA+, and in accordance with it, defines numbers, strings, Boolean operators (conjunction $\wedge$, disjunction $\vee$, negation $\neg$, implication $\Rightarrow$, equivalence $\Leftrightarrow$), sets, functions, records, tuples and sequences. For example, a set of tuples is described as the Cartesian product of the particular sets. Comments are denoted between "(*" and "*)" or after "*" ([Lamp09]). Symbols are typed as ASCII strings, e.g., "$\in$" is typed "\in", "$\wedge$" is typed "/\", "$\vee$" is typed "\/", etc. ([Lamp09]).

4.2.2 Architectures described in PlusCal

Before starting with the explanation of the PlusCal algorithms for architectures, the way of modelling spaces and the operations in XVSM are explained.

Spaces are simulated as lossy FIFO channels that use message sending and communicate synchronously. In both algorithms, operations *read*, *take* and *write* with FIFO coordinator and key coordinator are modelled by using the following operations on channels:

- *Head(s)* gives the first element of sequence *s,*
- *Tail(s)* gives the tail of sequence *s*, when its head is removed,
- *Append(s,e)* makes a new sequence that is obtained by appending element *e* to the tail of sequence *s,*
- *Len(s)* is the length of sequence *s*.

By using the numbered operations, macros for *read*, *take* and *write* with FIFO coordinator and *read*, *take* and *write* with key coordinator are made. Entries (i.e., tuples) in spaces are finite sequences. Therefore, modelling the operations with FIFO coordinator is self-descriptive. Modelling the operations with key coordinator implements the possibility to "extend" the channel, if needed. For example, macro `WriteKEY` allows for putting an element on 100th position in the channel, if the first three positions are with elements (so the length of the channel is 3), by assigning "emptiness" (<<>>) between 3rd and 100th position. Macros `TakeKEY` and `ReadKEY` use `await` statement, i.e., the actions can be done only in case that the length of a channel is greater or equal than the requested key (and that the channel itself is not empty). If so, then macro `ReadKEY` implies reading the element on the specified position, whereas macro `TakeKEY` implies taking the element from the specified position and changing the channel itself.

The next algorithm describes the first architecture - Lookup pattern composition:

```
--algorithm Lookup {
\* msgC-the array of channels for messages, spaces are simulated
\* as channels, i.e., msgC[1],...,msgC[N] simulate swarm spaces,
\* msgC[0] simulates the answer space
 variables msgC = [im \in 0 .. N |-> <<>>];
 define {
   clientNode == 0
   fromNode == 1
   currPath == 2
   pathPos == 3
   searchStr == 4
   swarmType == 5
   searchStatus == 6
   }
macro WriteFIFO (m , chan) { chan := Append(chan, m ) }
macro TakeFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan);
                              chan := Tail(chan)}
macro ReadFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan)}

macro WriteKEY (m , chan, key) { chan := [ikey \in 1 ..   (IF
Len(chan) < key THEN key ELSE Len(chan)) |-> IF ikey = key THEN
m ELSE IF ikey <= Len(chan) THEN chan[ikey] ELSE <<>>]}

macro TakeKEY  (v , chan, key) { await Len(chan) >= key /\
chan[key] /= <<>>; v:=chan[key];
chan := [ikey \in 1 .. Len(chan) |-> IF ikey = key THEN <<>>
ELSE chan[ikey]]}

macro ReadKEY  (v , chan, key) { await Len(chan) >= key /\
chan[key] /= <<>>; v := chan[key]};

process (Client = 0)
\* msg - the message with the request
variables msg = <<>>; initialPath = [ip \in 1 .. N |-> 0]; i; j;
{
 \* send M messages, i.e., requests for searching
  l1 : i := 1;
  l2 : while (i <= M) {
   \* nondeterministically select some node j
     with (rndNode \in 1 .. N) { j := rndNode;};
     \* the message goes to node j
     WriteFIFO (<<clientNode, initialPath, 0, "searchStr", "F",
                      FALSE>>, msgC[j]);
     i:=i+1;
     };
  \* wait for processing the request
   l3 : while (i > 1) {
    \* take the message with the processed request msg
    \* from its channel, i.e., from the "answer space" msgC[0]
       TakeFIFO (msg, msgC[clientNode]);
       i := i-1;
      };
 \* assert that there is nothing left in channels, i.e., that
 \* the number of sent messages minus the number of received
 \* messages equals to 0
   l5 : while (i <= N) {
           assert (Len(msgC[i]) = 0);
           i:=i+1;
         };
   assert (Len(msgC[clientNode]) = 0);
  }

process (Swarm \in 1 .. N)
variables msg = <<>>; nextNode; newPos; newType; newStatus; i;
          iNodes {
 l1 : while (TRUE) {
      either skip;
      or {
      \* accept the message with request msg from its channel
      \* msgC[self]
       TakeFIFO (msg, msgC[self]);
       \* processing ...
       \* if the type is forward
       if (msg[swarmType] = "F") {
       \* the path lenght is increased by one
       newPos := msg[pathPos]+1;
       \* newStatus simulates a search in a content space
       with (rndFound \in {TRUE, FALSE}) {newStatus:= rndFound};
       \* if new status is found or the path length equals N
       if(newStatus \/ newPos = N) {
       \* the type becomes backward
       newType := "B"
         };
       else {newType := "F"}
       };
       else {
       \* in "backward" case, the status of a search does not
       \*change and the current path length decreases by one
       newStatus := msg[searchStatus];
       newPos := msg[pathPos]-1;
       newType := "B";
       };
       \* selection of the next node
       if (newType = "F") {
       \* from set 1..N, exclude nodes that are in the path
       iNodes := 1 .. N \ {self};
       i := 1;
       l2 : while (i<newPos) {
          iNodes := iNodes \ {msg[currPath][i]};
          i := i+1;
          };
     \* from the rest of them, randomly (nondeterministically)
     \* select one, i.e., this simulates search and selection
     \* in the routing space
      with (rndNext \in iNodes) {nextNode := rndNext};
      };
      else {
     \* for the "backward" type, the value of a previous
     \* position in the path refers to the previous node
     \* or if it is in the first position, the processed
     \* message is sent (routed back) to the client
     if (newPos=1) {
     nextNode := clientNode;
      };
     else {
     nextNode := msg[currPath][newPos-1];
       };
      };
     \* the message is processed
     l3 : msg[fromNode] := self || msg[currPath][newPos] :=
          self || msg[pathPos] := newPos || msg[swarmType] :=
          newType || msg[searchStatus] := newStatus;
          WriteFIFO(msg, msgC[nextNode]);
      };
   };
 }
}
```

Listing 4.1. The Pluscal algorithm for Lookup pattern composition.

The constants are: *M* is the number of swarms with requests, *N* is the number of nodes. They are configurable parameters in the algorithm. As spaces are simulated as flexible channels, *msgC* denotes the array of channels for messages, i.e., *msgC*[1],...,*msgC*[*N*] simulate swarm spaces, whereas *msgC*[0] simulates the answer space. An entry (tuple) *msg* consists of the following information: <fromNode, currPath, pathPos, searchStr, swarmType, searchStatus>, where *fromNode* denotes the node from which the message is sent, *currPath* is the current path, i.e., the array of visited nodes, *pathPos* is the path position (the length of the current path), *searchStr* is the search pattern, *swarmType* is type F (forward) or B (backward), and *searchStatus* is the status of a search TRUE - found, FALSE - not found.

There are two processes: the client and the swarm agents.

The client (*Client*) is described as a process that sends *M* messages, i.e., requests for searching by writing them into load spaces. As `with` statement allows for nondeterminism, nodes in which load spaces will be put requests are selected randomly. Then the client waits for processing the requests and accepts the message with the processed request *msg* from its channel. At the end, the client asserts that the system works correct, i.e., the number of requests have entered in the system (input) is the same as the number of processed requests (output).

The swarm agents (*Swarm*) are described as a set of concurrent processes. A swarm agent accepts the message with request *msg* from its channel *msgC*[*self*], processes the request (it simulates the consulting of its content space) and according to the obtained feedback (true or false), it will continue: if TRUE or all nodes in the network are already visited, the status of the agent will be changed to "backward"; otherwise, the selection of the next node is done nondeterministically (it simulates the consulting of its routing space) and the request is routed to the swarm space of the next node. . In the `either` statement, `skip` is used to prevent waiting for "lazy" agents.

The next algorithm describes the second architecture - SILBA pattern composition.

```
--algorithm Silba {
\* msgC- the array of channels for messages, spaces are
\* simulated as channels, i.e., msgC[1],...,msgC[N] simulate
\*swarm spaces, msgC[0] simulates the answer space
\* allocC-the array of channels that simulate allocation spaces
\* nStatus–the array of nodes’ status

variables msgC = [im \in 0 .. N |-> <<>>]; allocC = [il \in 1 ..
N |-> <<>>]; nStatus = [in \in 1 .. N |-> "UL"];
define {
  clientNode == 0
  fromNode == 1
  reqID == 2
  partnerNode == 3
  }

macro WriteFIFO (m , chan) { chan := Append(chan, m ) }
macro TakeFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan);
                              chan := Tail(chan)}
macro ReadFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan)}
```

```
macro WriteKEY (m , chan, key) { chan := [ikey \in 1 ..   (IF
Len(chan) < key THEN key ELSE Len(chan)) |-> IF ikey = key THEN
m ELSE IF ikey <= Len(chan) THEN chan[ikey] ELSE <<>>]}

macro TakeKEY  (v , chan, key) { await Len(chan) >= key /\
chan[key] /= <<>>; v:=chan[key];
chan := [ikey \in 1 .. Len(chan) |-> IF ikey = key THEN <<>>
ELSE chan[ikey]]}

macro ReadKEY  (v , chan, key) { await Len(chan) >= key /\
chan[key] /= <<>>; v := chan[key]};

process (Client = 0)
\* msg - the message with the request
 variables msg = <<>>; i; j; {

  \* send M messages, i.e., requests
  l1 : i := 1;
  l2 : while (i <= M) {
       \* select randomly (nondeterministically) some node j
       with (rndNode \in 1 .. N) { j := rndNode;};
       \* the message goes to the msg channel of node j, i.e.,
       \* to the "load space" of node j
       WriteFIFO (<<clientNode, i, clientNode>>, msgC[j]);
       i:=i+1;
       };
   \* wait for processing the request
   l3 : while (i > 1) {
        \* accept the message with the processed request msg
        \* from its channel msgC[0], i.e., msgC[0] refers to
        \* the answer space
        ReadKEY (msg, msgC[clientNode], i-1);
        i:=i-1;
        };
   \* assert that there is nothing left in channels,i.e., that
   \* the number of sent messages minus the number of received
   \* messages equals to 0
   l4 : while (i <= N) {
        assert (Len(msgC[i]) = 0 /\ Len(allocC[i])=0);
        i:=i+1;
        };
   assert (Len(msgC[clientNode]) = M);
 }

process (WA \in 1 .. N)
```

```
variables msg = <<>>; {
  l1 : while (nStatus[self] /= "OL") {
       either skip
        or {
        \* accept the message with the request msg from its
        \* channel msgC[self], i.e., "load space"
         TakeFIFO (msg, msgC[self]);
         \* processing ...
          skip; \* "execute" some job
         l2 : msg[fromNode] := self;
             WriteKEY(msg,msgC[clientNode],msg[reqID]);
         };
   l3 : if (Len(msgC[self]) < T1) {
         nStatus[self] := "UL"
         }
         else {
          if (Len(msgC[self])<T2) nStatus[self]:= "OK";
          else nStatus[self] := "OL"
          }
        };
   }

process (Arbiter \in 1 .. N)
variables msg = <<>>; {
  l1 : while (nStatus[self] = "OL") {
       either skip
       or {
       \* accept the message with the request msg from its
       \* channel msgC[self], i.e., "load space"
        TakeFIFO (msg, msgC[self]);
        \* send the request to allocC, i.e., "allocation space"
      l2 : msg[fromNode] := self;
           WriteFIFO(msg, allocC[self]);
        };
      };
 }

process (RA \in 1 .. N)
variables msg = <<>>; ack; i; pNodes; pNode; {
  l1 : while (TRUE) {
       either {
        if (nStatus[i]="UL") {
        i := 1;
        pNodes := 1 .. N;
      l2 : pNodes := pNodes \ {self};
      l3 : while (i<=N) {
        if (nStatus[i]/="OL") pNodes:=pNodes\ {i};
        i := i+1;
           };
           with(rndNode \in pNodes) {pNode := rndNode};
           WriteFIFO(<<self,0,pNode>>, allocC[self]);
           };
         };
         or {
         \* accept the message with the request msg from its
         \* channel allocC[self], i.e.,  "allocation space"
          ReadFIFO (msg, allocC[self]);
          if (msg[partnerNode] = clientNode) {
          \* find UL or OK node
          i := 1;
          pNodes := 1 .. N;
        l4 : pNodes := pNodes \ {self};
        l5 : while (i<=N) {
          if(nStatus[i]="OL")pNodes:=pNodes\ {i};
            i := i+1;
           };
          with (rndNode \in pNodes) {pNode := rndNode};
          allocC[self][1][partnerNode] := pNode;
          };
       };
     };
 }

process (OUTag \in 1 .. N)
variables msg = <<>>; pNode; {
  l1 : while (TRUE) {
       either skip
       or {
       \* accept the message from the channel
        ReadFIFO (msg, allocC[self]);
       if (msg[partnerNode] /= clientNode /\ msg[reqID]>0) {
       \* take the message from allocC and write it to the
       \* "load space" of pNode
     l2 : TakeFIFO (msg, allocC[self]);
           pNode := msg[partnerNode];
    l3: msg[fromNode] := self || msg[partnerNode] := clientNode;
          WriteFIFO(msg, msgC[pNode]);
         };
      };
   };
 }


process (INag \in 1 .. N)
variables msg = <<>>; pNode; {
  l1 : while (TRUE) {
       either skip
       or {
       \* accept the message from the channel
        ReadFIFO (msg, allocC[self]);
        if (msg[partnerNode] /= clientNode /\ msg[reqID]=0) {
        \* take the message from allocC and write it to the
        \* "load space" of pNode
    l2 : TakeFIFO (msg, allocC[self]);
           pNode := msg[partnerNode];
    l3 : TakeFIFO(msg, msgC[pNode]);
    l4: msg[fromNode] := self || msg[partnerNode] := clientNode;
         WriteFIFO(msg, msgC[self]);
         };
      };
   };
 }
}
```

Listing 4.2. The PlusCal algorithm for SILBA pattern composition.

The constants are: *M* is the number of requests (tasks), *N* is the number of nodes, T1 and T2 are the threshold levels in the transfer policy and they are configurable parameters in the algorithm. As spaces are simulated as flexible channels, *msgC* denotes the array of channels for messages, i.e., *msgC*[1],...,*msgC*[*N*] simulate load spaces, whereas *msgC*[0] simulates the answer space. An entry (tuple) *msg* is simplified and consists of the most important information: <fromNode, reqID, partnerNode>, where *fromNode* denotes the node from which the message is sent, *reqID* is the identification of a request (i.e., message), and *partnerNode* identifies the most suitable node for exchanging the request.

There are six processes: a client, worker agents, allocation agents (arbiter agents, IN agents, OUT agents), and routing agents.

The client (*Client*) is described as a process that sends *M* messages, i.e., requests for searching by writing them into load spaces (FIFO coordinator is used). As `with` statement allows for nondeterminism, nodes in which load spaces will be put requests are selected randomly. Then the client waits for processing the requests and accepts the message with the processed request *msg* from its channel (KEY coordinator is used). At the end, the client asserts that the system works correct, i.e., the number of requests has entered in the system by clients (input) is the same as the number of processed requests (output).

The worker agents (*WA*) are described as a set of concurrent processes. If the current node is not overloaded, the worker agent accepts the message with the request *msg* from its channel *msgC*[*self*], i.e., from the load space of its node, where *self* denotes the identifier of the process itself. Further, the worker agent will execute a job. In the algorithm, the execution is skipped (statement `skip`) as it depends on a particular type of a job (e.g., compile tasks, etc.) and is not of interest for the architecture itself. Finally, the simulation of the transfer policy is described, so the status of a node is updated dynamically. In the `either` statement (the description of the worker agent and all other agents), `skip` is used to prevent waiting for "lazy" agents.

The allocation agents (arbiter, IN, OUT) are further described.

The arbiter agents (*Arbiter*) are described as a set of concurrent processes. If the current node is overloaded, the arbiter accepts the message with the request *msg* from its channel *msgC*[*self*], i.e., from the load space of its node and sends, i.e., writes the request to the allocation space of its node.

The OUT agents (*OUTag*) are described as a set of concurrent processes that accept the message with the request from the channel. If the request is "already known", i.e., identified in the current load space, it takes the message from the allocation space and writes it to the load space of its partner node (*pNode*). The IN agents (*INag*) are described similarly. In this case, if the request is not "already known", i.e., identified in the current load space, it takes the message from the allocation space and writes it to its load space.

The routing agents (*RA*) are also described as a set of concurrent processes. As routing agents are "in charge" for realization of the location policy, the finding of the partner node is abstracted in the algorithm's description and a partner node is chosen nondeterministically - it simulates the consulting of the routing spaces and performing of a particular load balancing algorithm.

All atomic actions are labelled in the algorithm. Assignment statements separated by | | form a multi-assignment, executed by first evaluating all the right hand expressions and then performing all the assignments.

TLA+ specifications of architectures are made (Appendix B) and correctness of them is proven via TLC model checker. So, both architectures are correct, independently of algorithm(s) and a network topology used.

4.3 Implementation

According to these specifications of architectures and consequently, their verification, the implementation is done in Java, and the benchmarks are run (chapter 6). As already stated - spaces are simulated as flexible channels that use message sending and communicate synchronously. However, the usage of autonomous spaces with an asynchronous communication is a more general way to realize it. Therefore, as a model for asynchronous, autonomous communication, XVSM is used (chapter 2, subsection 2.1.5). XVSM is middleware technology that allows high decoupling and the access to the share data. It serves for autonomous acting agents that communicate in the environment in a P2P distributed way. The information can be put somewhere in a space by an agent and the other agent can pick it up. Therefore, XVSM perfectly fits to all described requirements.

4.3.1 LookUp Implementation

A space-based architecture is used for the implementation. The swarm space is a container with a FIFO coordinator, whereas the answer space is a container that uses a key coordinator. Fig 4.8. shows a local node agent interaction. Clients issue request by writing a tuple like "[**searchPattern**: myURL, **params**:myProg, **nodeID**:myID, **visitedNodes**:myList, **clientID**:135, **reqID**:246, **timeout**:100]"into the swarm space. A swarm agent consults a content space and in case that the desired content is found, writes the answer like "[**result**:resultURL, **visitedNodes**:myList]" into the answer space. Otherwise, it consults the routing space and chooses the next node for searching.

Fig 4.9. represents an interaction scenario of a FORWARD ant agent that has not found a searching information on node A. Actually, it represents just a segment (routing information from node A to node B) of the complete routing scenario, as it continues in the same way from one node to another until one of the resulting situations is reached: exact data found on some node X, acceptable data found on some node X or no data found. A swarm agent reads the information from the content space, i.e., consults the content space trying to find a specified content. In case that the desired content is not found, a swarm agent reads routing information from the routing space, based on that decides which node to visit next and transfers the request from the current site to the next site.

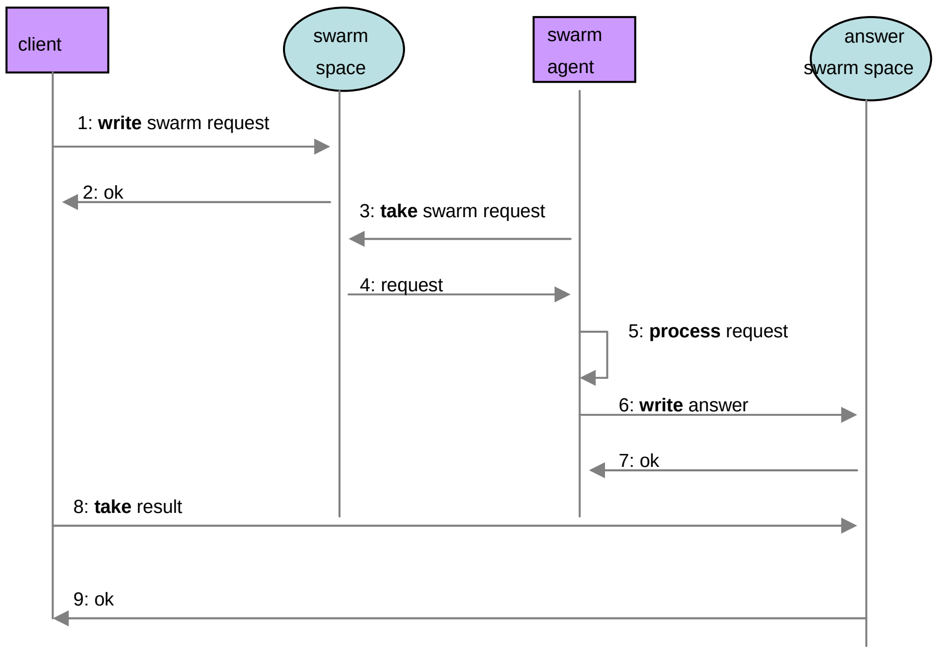


Figure 4.8. Local Node Implementation

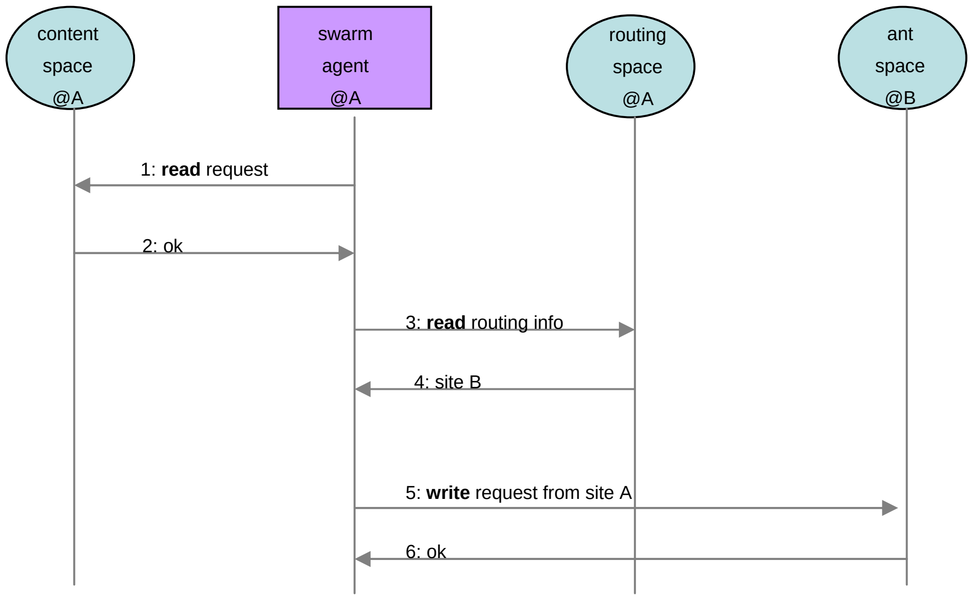


Figure 4.9. Forward Ant Implementation

4.3.2 SILBA implementation

Local Node Implementation

A load space is a container with an implicit coordinator, so-called load coordinator that keeps track of every request and workers registrations, and implements a transfer policy[10]. The answer space uses a key coordinator.

Fig. 4.10. shows a scenario of agent collaboration at local node level. First, each worker agent must register at the load space. Then clients can issue requests into the load space. The worker agents compete for tasks but only one will be able to execute the take operation using a new local transaction, execute the task, write the result as answer entry into the answer space using the transaction, and finally commit the transaction. If a worker that fails after having called take request and before committing the transaction, the timeout given at transaction start will fire and cause the rollback of the transaction. Failover is achieved in that another worker can take the request. Finally, the client takes the result from the answer container, correlating it with its request via the request ID, using the key coordinator for that.

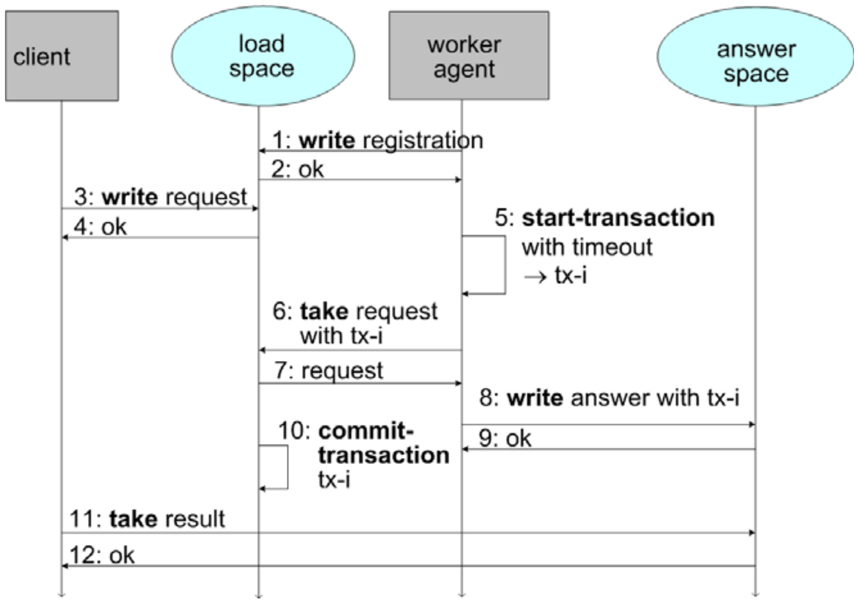


Figure 4.10. Local Node Implementation ([KüŠe09])

[10] As the coordinator is pluggable, the transfer policy can be changed at any time, even dynamically.

Allocation Implementation

Allocation implementation is described by using an example that shows an interaction scenario of an OUT allocation agent (Fig. 4.11). The allocation agent reads the load information from the load space, and writes this information into the routing space. If the result is OL, it generates an OUT routing request. The OUT agent watches for outgoing routing requests, in a newly created transaction takes a next routing request, tries to read a partner information from the allocation space, and if found, takes *k1* (*k1* ≤ *k*) requests from its load space and transfers them to the found partner site, and finally commits its local transaction[11].

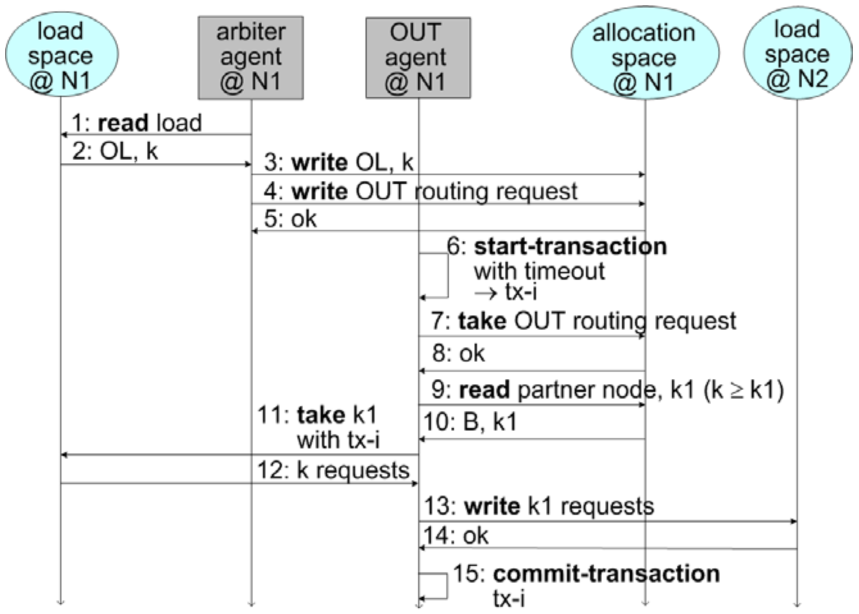


Figure 4.11. Allocation Implementation ([KüŠe09])

[11] If no outgoing routing request is found, or if not (yet) partner information exists, it will abort the transaction and try later. For the case that the worker crashes, a transaction timeout is used at transaction start.

Routing Implementation

Routing implementation describes how the location policy that resolves requests for partner nodes. It is explained by continuing and extending the example from the allocation implementation. Next figure (Fig. 4.12.) depicts a basic routing scheme started by node A. An OUT routing request is found in the allocation space of node A. The routing agent at node A reads this request from its allocation space and reads the routing information from its routing space, and routes the request to the neighbour(s), e.g. node B. The routing agent at site B behaves in the same way, repeats the explained procedure, and this goes further until a routing agent at a node X finds out that its local node is OK or UL, and therefore, can accept a certain amount *k1* (*k1* ≤ *k*) of requests. It will send the feedback (write this information back) directly in a P2P way to the originally requesting site A.

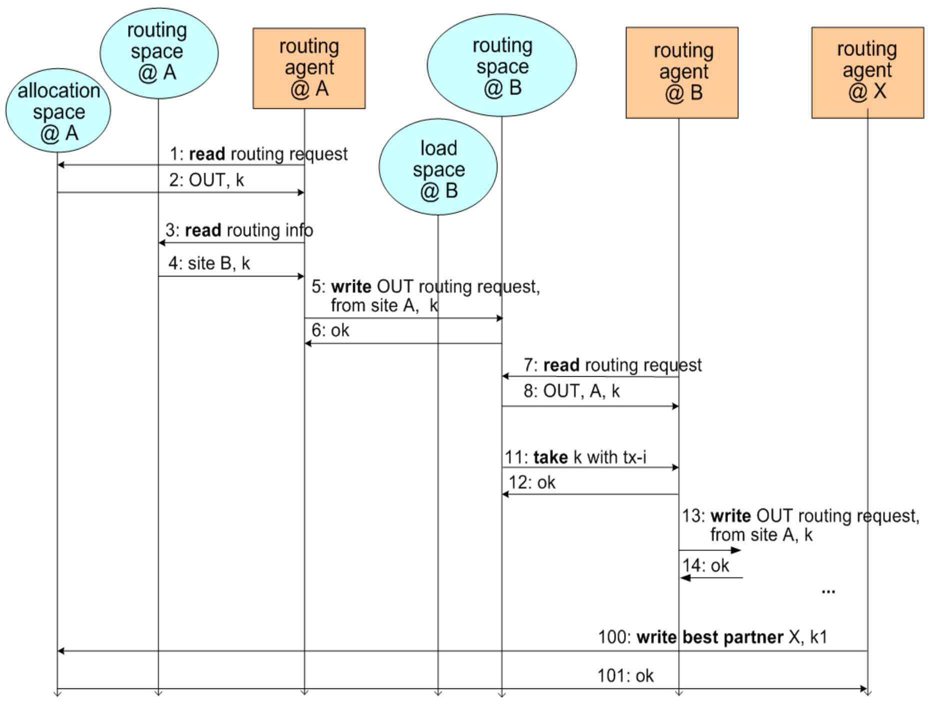


Figure 4.12. Routing Implementation ([KüŠe09])

Implementation Parameters

In the implementation, we introduced one parameter, so-called *search mode* that is configurable and determines which nodes in the network (according to their load status) will trigger a load-balancing algorithm.

Table 4.2. Search Modes ([ŠeKü10c])

SM1	the algorithm is triggered from UL nodes, OK nodes (in a situation when it's likely that the node will become OL, but is not yet heavily loaded) and consequently OL nodes.
SM2	the algorithm is triggered from UL nodes
SM3	the algorithm is triggered from OK nodes (in a situation when it's likely that the node will become OL, but is not yet heavily loaded) and consequently OL nodes; the computation of x argument for *δ(x)* suitability is slightly changed[12].
SM4	the algorithm is triggered from OL nodes
SM5	the algorithm is triggered from UL and OL nodes
SM6	the algorithm is triggered from OK nodes (in a situation when it's likely that the node will become OL, but is not yet heavily loaded) and consequently OL nodes.

For *suitability* function *δ*, we implemented the following functions:

Table 4.3. Suitability Functions ([ŠeKü10c])

SF0	one linear function: if $(x = 1.0)$ $\delta(x) = n$, else $\delta(x) = 5x$ (if the number of nodes $\leq n$)
SF1	an exponential function: $\delta(x) = 10^x$
SF2	a polynomial function: $\delta(x) = 10x^3$
SF3	another linear function: if $(x < 1.0)$ $\delta(x) = 4nx$, else $\delta(x) = 5n$ (if the number of nodes $\in [5n\text{-}4, 5n]$)

[12] If a node is in OK state, the algorithm is triggered and searching for a suitable node among the neighbour nodes is started (afterwards, this information about the most suitable node is stored locally). As soon as the node gets OL, the tasks get re-routed to this target node. To achieve this “a priori” searching for a suitable node (when the information about a task is still unavailable, i.e., the task complexity *c* is yet unknown), we computed argument *x* only on the basis of host speed and host load parameters.

The *fitness* function *f* is computed from the suitability function of the found node and the number of hops to this node and we used the following combinations:

Table 4.4. Fitness Functions ([ŠeKü10c])

FF0	$f(x) = \delta(x)$ / number_of_hops
FF1	$f(x) = \delta(x)$ • (quality_of_links / number_of_hops)
FF2	$f(x) = \delta(x)$ / sqrt(number_of_hops)
FF3	similar to FF0, only the local node is excluded from the comparison and the rest of neighbouring nodes are taken in consideration.

4.4 Summary

In this chapter, two self-organizing coordination architectures on the pattern layer were developed: architecture for searching, retrieving and placing information in the Internet and SILBA (which stands for self initiative load balancing agents) for load-balancing in heterogeneous distributed systems. In case of information retrieval scenario, a new overlay network with an intelligent lookup mechanism is developed and implemented in this thesis. The chosen overlay is a purely decentralized and unstructured one. It allows for the plugging of the intelligent lookup mechanism is based on swarm intelligence (that is described in chapter 5), able to navigate successfully through the network of data and scales well. A new generic architectural pattern SILBA is proposed and developed for a load balancing. It allows for the plugging of different load balancing algorithms, and can be composed towards arbitrary network topologies. First, the basic SILBA composed of several sub-patterns is described. Further, SILBA is extended on several layers that allow routing between different subnets, simultaneously with load balancing between nodes within these subnets.

Both architectures represent self-organized decentralized and decoupled models in which different types of autonomous agents work concurrently and continuously, collaborating through the spaces. The different patterns are the parts of the complete architecture. The architectures are modeled and communication in patterns is described by using PlusCal algorithm language which is meant to replace pseudo-code for writing high-level descriptions of algorithms. The architectures are justified (via TLC model checker) and it is proven that they are correct for all combination of algorithms that can be plugged in, all policies and all topologies.

CHAPTER 5

5 EMPLOYING NATURE-BASED MECHANISMS

Nature is prolific with self-* mechanisms. Learning from nature and applying this knowledge in solving IT problems is proven to be beneficial. In this chapter, the emphasis is put on *swarm intelligence*:

- Ant algorithms have been used in applications, but still not enough exploited and therefore offer a challenge of improving solutions in the application scenarios (Chapter 3).
- Bee algorithms are relatively new and although some applications exist, they are still developing, and have neither a general form nor a theoretical basis established; according to some specific features that bee intelligence offers, their adaptation towards the applications in the presented scenarios (Chapter 3) is very attractive.

This chapter starts with a general comparison of Ant Algorithms versus Bee Algorithms, continues with the description of these algorithms and further, the ways of mapping/adapting these algorithms to the application scenarios are described.

5.1 Swarm Intelligence

Different dynamic processes characterize the application scenarios. According to the characteristic of P2P networks (Chapter 2), it is already explained that in an unstructured P2P network, a placement of information can be done independently of an overlay topology, but the content must be localized explicitly. Unstructured P2P networks fit better to the considered scenarios as they support better dynamical processes from both scenarios, like: nodes can dynamically join and leave, the information about load changes permanently, tasks are dynamically added and continuously processed, queries that are more complex are possible, etc. Algorithms used in a particular overlay network are inspired by swarm intelligence. Ant algorithms are inspired by behaviour in an ant colony and have some applications up to now (Chapter 2). Bee algorithms are inspired by behaviour of a honey bee colony and represent a relatively new application of one bio-mechanism with a limited number of applications and without theoretical basement. As both algorithms belong to swarm intelligence, the main issue that differentiates them in the scope of their mapping to IT problematic is presented first.

Bees communicate directly with their hive mates, i.e., if a bee finds the required information, it flies back to its hive and informs the “starting place” directly in a P2P way. So, knowledge distribution takes place in the own hive. Bees of different hives do not communicate with each other, and bees that are out of their hive do not communicate with other bees.

Ants communicate indirectly with their nest mates. They leave information (pheromones) at all nodes on the backward trip. Their forward trip is comparable to the bees' forward movement (navigation), but the backward trip is different – the ant does not directly contact the "starting place" in a P2P way but must go the entire way back.

5.2 Ant Algorithms

5.2.1. Ant behaviour in nature

Ants represent a collective intelligence in nature. Ant colonies consist of individual ants with simple behaviour and limited cognitive abilities. So, they are not capable to solve complex problems individually. In spite of that, they are highly structured social organization, capable of solving complex tasks at the collective level, such as constructing optimal nest structure, or finding the shortest path between a food source and their nest. Building of chains of ants ([LSTD01]), formation of drops of ants ([TBSDL01]), brood sorting, cooperative transport are only some of their observed behaviours.

One of these mechanisms - finding the shortest path between a food source and their nest – served as an inspiration for interdisciplinary self-organization researches in order to be mathematically modelled and described, and later shaped in an algorithmic form for IT usage. The amazing results from nature, observed in a convergence to the shortest path, is their everyday simple activity based on pure random movements, local decisions, fully distributed, autonomous and adaptive process. In nature, ants wander randomly, seeking for food. Once when they find food, they return to their nest laying down pheromone that forms an evaporating chemical path. It is a form of indirect communication mediated by modification of the environment, so-called stigmergy. Thus, ants highly coordinate their behaviour and activity via stigmergy. Using this indirect communication, other ants locate this trail, follow it and reinforce it, since they also lay down pheromone. As a result, shorter paths to food have more pheromone and are more likely to be followed. Thus, this positive feedback eventually leads all the ants following a single path.

In order to investigate behaviour of ants, ([DAGP90]) performed the so-called "double-bridge experiments". In the first experiment, ants had possibility to choose between two paths of the same length between their nest and the food source. At the very beginning, there were no pheromones on the paths, so ants chose the path randomly. Both paths had the same probability to be chosen. After some time, due to the random fluctuation, there were a few more ants on one path. As a consequence, there were more pheromones on that path that became more and more reinforced. So, the ants converged to one path. This is the consequence of positive feedback in a self-organizing behaviour of the ants. In the second experiment, the lengths of paths were different. One path was twice as long as the other. Again at the beginning, without pheromone on the paths, both paths seemed to be identical to the ants. After a while, the pheromone logically started faster to accumulate on a shorter path, making consequently this path more attractive for the ants. In this situation, the convergence of the ants to a shorter path was driven mainly because of the differential path length.

5.2.2. Algorithms

The original idea for the ant colony optimization metaheuristic (ACO) comes from nature, observing the behaviour of real ants and their search for food. ACO algorithms are probabilistic techniques for solving computational problems that are based in finding as good as possible paths through graphs by imitating the ants' search for food. Mapping to the artificial ant colony where a software agent plays the role of an ant, ACO supposes a multi-agent organization. The natural pheromone is stigmergic information that serves for the communication among the agents. Ants make pure local decisions and work in a fully distributed way. From the basic Ant System algorithm ([DoSt05], [MMBR09]), different variations and extensions are derived like: Elitist Ant System, Rank-Based Ant System, Min-Max Ant System (MMAS), and Ant Colony System ([DoSt05]). The most popular applications of ACO algorithms are: (network) routing, assignment, scheduling, machine learning, etc. ([DoSt05]). Also, ACO can be combined with other algorithms, e.g., genetic algorithms ([PoMe08]), forming hybrid algorithms. All of them consider a static scenario (adding and removing the network components are not supported). AntNet ([DiDo98a]) is a network routing algorithm (originally constructed for adaptive routing in IP networks), based on Ant Colony Optimization (ACO). This algorithm considers a dynamic scenario - it supports adding and removing the network components, is highly adaptive to network and traffic changes, and robust to agent failures. A detailed description of these algorithms can be found in ([DiDo98a]) and ([DoSt05]).

In the following, the next algorithms are described: a basic Ant System as the underlying algorithm, and Min-Max Ant System and AntNet as they are used in this dissertation.

The behaviour of real ants is first mathematically described by means of a stochastic model. Afterwards this model is expressed by using graphs theory.

Let us consider a static, connected graph $G = (C,L)$, where C is the set of n nodes and L is the set of undirected arcs connecting them. Two nodes, $i,j \in C$, are neighbours, if there exists an arc $(i,j) \in L$.

Ant System algorithms ([DoSt05]) consist of two phases: the ants' tour (solution) construction and the pheromone update.

Phase**1** - A tour construction: *m* artificial ants concurrently build their solutions. Initially, they are positioned on randomly chosen nodes, start their trips and choose the next node to be visited in their trip(s) by applying a random proportional rule[13] ([DoSt05]):

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \text{ if } j \in N_i^k \tag{5.1}$$

where τ_{ij} is a pheromone trail on (i,j)-arc, $\eta_{ij} = 1/d_{ij}$ is a heuristic value (available à priori), α and β are two parameters that determine the influence of the pheromone trail and the heuristic information, and N_i^k is the set of nodes from the neighbourhood of node i that ant k has not visited yet.

Phase **2** - Pheromone update: A pheromone value on all arcs is decreased (after all ants finished phase1) by a constant factor ([DoSt05]):

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} \tag{5.2}$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate. After evaporation, the additional amount of pheromone is deposited on the arcs that have been crossed in the ants' constructions of solutions, i.e., that have been used in phase1:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{5.3}$$

where $\Delta\tau_{ij}^k$ is the amount of pheromone that ant k deposits on arcs it has visited.

MMAS Ant System is one popular and successful improvement of the initial Ant System algorithm. It includes the following modifications ([DoSt05]):

[13] This rule is a derived from the basic statistics' rules and represents the probability of choosing the path, i.e., arc (i,j) when being located at the node i, where j is one of the nodes from the neighbourhood of node i.

- A strong exploitation of the best tours found in order to accelerate the convergence of the process,
- The possible range of pheromone trail values are limited to the interval [τ_{min}, τ_{max}], where τ_{min} and τ_{max} are the lower and upper limits respectively on the possible pheromone values introduced in order to avoid search stagnation that could happen due to the premature convergence.
- The pheromone trails are initialized with the upper pheromone trail limit,
- The pheromone trails are reinitialized (dynamically calculated) each time the system approaches any kind of stagnation (on this way, a higher exploration of solutions is forced).

The first phase is the same as in the initial Ant System algorithm, but the second phase is modified – the update of pheromone trails is implemented as follows ([DoSt05]):

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best} \tag{5.4}$$

where $\Delta\tau_{ij}^{best} = 1/C^{best}$ and C^{best} can be either the length of the iteration best tour (i.e., the best solution in the current iteration) or the length of the best-so-far tour (i.e., the best solution from the beginning of the trial).

AntNet algorithm ([DiDo98a]) uses the same algorithmic pattern as the other algorithms from Ant System, i.e., it also has two phases: a solution construction and data structures[14] update. The characteristics of this ACO algorithm is that it is specifically constructed for data network routing. Routing is the process of selecting paths in a network along which to send network traffic and it "encompasses" distributed activities of building and using routing tables. Routing table, stored in a node (networked computer), lists the routes to particular network destinations, and in some cases, metrics associated with those routes. A routing table is maintained by each node in the network and it contains information about the topology of the network immediately around the node (i.e., it contains information important for making local forwarding decisions). As it is expectable to have a high fluctuation of data in a network, the nodes and links suddenly can be broken (or can be added), this algorithm supports a high dynamics.

Specific types of data structures are introduced: an artificial pheromone matrix, T_i and a statistical model, M_i of the traffic situation over the network. Both matrices are associated with node i of the data network. The artificial

[14] Data structures are described below in the text.

pheromone matrix, T_i has the elements τ_{ijd} that describe the learned desirability for an ant in node *i* with destination *d* to move to node *j*. The statistical model, M_i of the traffic situation serves to evaluate the paths produced by the ants. The model is described as $M_i(\mu_{id}, \sigma_{id}^2, W_{id})$ where μ_{id} is the sample mean, σ_{id}^2 is the variance, W_{id} is the "observation window" used to store the best value $W_{best_{id}}$ of the ants' trip time from node *i* towards destination *d*. The sample mean and variance provide the expected time to go from node *i* to node *d* and are calculated by using the exponential models ([DiDo98a]):

$$\begin{aligned} \mu_{id} &\leftarrow \mu_{id} + \varsigma(o_{i\to d} - \mu_{id}) \\ \sigma_{id}^2 &\leftarrow \sigma_{id}^2 + \varsigma((o_{i\to d} - \mu_{id})^2 - \sigma_{id}^2) \end{aligned} \tag{5.5}$$

where $o_{i\to d}$ is the new ant' trip time from node *i* to destination *d* and ς is a real parameter that weighs the number of the most recent samples that will really affect the average. Matrix *M* maintains absolute distance/time estimates to all nodes, whereas matrix *T* gives the measure of relative goodness for each link-destination pair ([DiDo98a]).

Two sets of artificial ants exist: forward ants and backward ants. They differ according to their "actions":

- Forward ant, $F_{s\to d}$, travels from source node *s* to destination node *d*.
- Backward ant, $B_{s\to d}$, travels back to source node *s* by using the same path as $F_{s\to d}$ but the opposite direction; it uses the information collected by $F_{s\to d}$ in order to update routing tables of the visited nodes.

Each $F_{s\to d}$ starts its travel from the source node *s* and chooses its destination *d* according to this probabilistic rule[15] ([DiDo98a]):

$$p_{sd} = \frac{f_{sd}}{\sum_{i=1}^{n} f_{si}} \tag{5.6}$$

where f_{xy} is a some measure of data flow $x \to y$ and *n* is the number of nodes.

[15] This rule is also a derived from the basic statistics' rules with a similar pattern as Eq.4.1. and represents the probability of creating at node *s* a forward ant with node *d* as destination.

Phase**1** – A solution construction: The ant constructs the path on this way:

a) An ant that is currently at node *i* chooses the next node *j* to be visited by applying the following probabilistic rule ([DiDo98a]):

$$p_{ijd} = \frac{\tau_{ijd} + \alpha\eta_{ij}}{1 + \alpha(|N_i| - 1)} \tag{5.7}$$

where τ_{ijd} is an element of the pheromone matrix τ_i that indicates the learned desirability for an ant in node *i* with destination *d* to move to node *j*, $|N_i|$ is the number of neighbours of node *i*, η_{ij} is a heuristic value that takes into account the state of the j^{th} link queue[16] of the current node *i* ([DiDo98a]):

$$\eta_{ij} = 1 - \frac{q_{ij}}{\sum_{l=1}^{|N_i|} q_{il}} \tag{5.8}$$

The parameter *α* (from Eq.5.6.) weighs the importance of the heuristic values with respect to the pheromone values stored in the pheromone matrix.

b) When $F_{s\to d}$ comes to destination node *d*, it generates $B_{s\to d}$, then it transfers to $B_{s\to d}$ all of its memory and is being deleted.

c) $B_{s\to d}$ travels back to the source node *s* using the same path as $F_{s\to d}$ but the opposite direction. It uses the information collected by $F_{s\to d}$ in order to update routing tables of the visited nodes.

Phase **2** – Data structures update: This phase considers updating matrices T_i and M_i by the backward ant. It also refers to updates of entries corresponding to every node $d' \in S_{i\to d}$ (where $S_{i\to d}$ is a memory stack and $d' \neq d$) on the "subpaths" followed by ant $F_{s\to d}$ after visiting the current node *i*. In the pheromone matrix, T_i, those values that suggest choosing neighbour *f* when destination is *d* are incremented ([DiDo98a]):

$$\tau_{ifd'} \leftarrow \tau_{ifd'} + r \cdot (1 - \tau_{ifd'}) \tag{5.9}$$

where *r* is a value used by the backward ant $B_{s\to d}$ traveling from node *f* to node *i*. The intention is to increase the pheromone values $\tau_{ijd'}$ proportionally - small

[16] The traveling information is divided into data and routing "packets". Every node has a capability to "store-and-forward" by using a buffer space (as a shared resource among all the queues) for the incoming information and outgoing data.

pheromone values are increased proportionally more than large pheromone values. It implies a quick exploitation of new and good discovered paths.

The values of pheromones $\tau_{ijd'}$ for destination *d'* of the other neighbouring nodes $j, j \in N_i, j \neq f$, are reduced in a way that the sum of pheromones on links exiting from node *i* will remain 1:

$$\tau_{ijd'} \leftarrow \tau_{ijd'} - r \cdot \tau_{ijd'} \quad j \in N_i,\ j \neq f \tag{5.10}$$

There are several ways to determine and assign *r* values: from the simplest way of setting *r = constant* to more complex way that defines *r* as a function of the ant's trip time and parameters of the statistical model M_i ([DiDo98a]):

$$r = C_1 \frac{W_{best}}{T} + C_2 \frac{I_{\sup} - I_{\inf}}{(I_{\sup} - I_{\inf}) + (T - I_{\inf})} \tag{5.11}$$

where I_{sup} and I_{inf} are estimates of the limits of an approximate confidence interval μ and W_{best} is the best ant's trip time, C_1 and C_2 are configurable parameters that weigh the influence of the two parts in Eq. 5.11: the first part expresses the ratio between the best trip time observed over the current observation window and the current trip time, whereas the second part gives an estimation how far the value *T* is from I_{inf} in relation to the extension of the confidence interval ([DiDo98a]).

Figure 4.2. represents a pseudo-code of ACO metaheuristic: *constructSolution* corresponds to phase1 (i.e., ants *construct solutions* by moving from the origin to the destination, step-by-step, according to a stochastic decision policy); *pheromoneUpdate* corresponds to phase2 (i.e., the aim of *pheromone update* is to increase the pheromone values associated with good solutions (*deposit pheromones*) and decrease those associated with *bad ones); deamonActions* denotes the optional actions that cannot be preformed by single ants (like the application of a local optimization procedure).

```
procedure ACO_MetaHeuristic
  while(not_termination)
        constructSolutions()
        pheromoneUpdate()
    daemonActions()
  end while
 end procedure
```

Figure 5.1 Pseudo-code of ACO metaheuristic ([DoSt05])

Ant Algorithms are theoretically partially based. The convergence proofs do not apply to the ACO metaheuristic generally ([DoSt05]), but only to one subset of ACO algorithms (more specifically, MMAS Ant System and Ant Colony System).

5.2.3. Mapping Ant Algorithms to Application Scenarios

In this subsection, the mapping of ant algorithms to the considered scenarios is described. It is already stated that the considered scenarios contain NP-hard optimization problems. Ant algorithms as the type of promising meta-heuristics are applied. As the scenarios have their specificities, the ant algorithms are adapted and remodelled.

Ant Algorithms for Information Retrieval

The algorithms use autonomous agents technology and are inspired by swarm intelligence. Software agents act in swarms, i.e., more specifically, each software agent performs the role of an artificial ant. One of the natural multi-agent systems, ant colony, is fully distributed, self-organizing, with a high level of autonomy. Thus, a couple of ant algorithms are adapted for this use case. Lookup containers "communicate" with each other through a coordination space. In order to locate a container (to find its URL for a given published name) in an unstructured system, a nondeterministic search is applied. Searching through the network and the complete lookup mechanisms is realized by using the artificial ants: a forwarding ant carries a search request and a backwarding ant possesses a reply to the search request. Each lookup container "forwards" the request to its neighbours, i.e., it sends the forward ant that carries the request. The ants are randomly positioned (on containers) and the whole process is done concurrently. The responses are routed back by backward ants. The above mechanism is implemented by assigning each ant a published name and equipping each lookup container with published names and URLs.

- **Writing of information:** for placing the content into the network, two ways are used:
  randomly – the content is put randomly and there is no need for swarm intelligence algorithms,

"*brood sorting*" ([CMVT07a]) – this simulates brood sorting mechanism in ant colony from nature; entries are distributed on the basis of their type (similar entries stay closer to each other).

- **Retrieving of information:** For the case of lookup and data retrieving, two different ant algorithms are implemented: an adaptation of MMAS (suitable for a static scenario) and an adaptation of AntNet (suitable for a dynamic scenario). MMAS is implemented in a hybrid form, i.e., combined with Local Search. The search space is the set of URLs. The environment is static concerning the number of lookup containers (a finite number of them), but dynamic concerning the connection between lookup containers. The extension can be done on the basis of properties of AntNet algorithm by allowing a dynamic environment (i.e., the number of lookup containers can vary). An adaptation of ant algorithms comprises the following changes:

In the procedure c*onstructSolution* ([DoSt05]), the random proportional rule is used (see Eq.4.1.). The heuristic values from this rule are interpreted as a quality of the used links[17], expressed in time needed for an ant to traverse a particular path from lookup container A to lookup container B by using a particular link. The estimated length of the optimal tour (an initialization phase in MMAS algorithm) is the length of a tour generated by the nearest - neighbour algorithm[18] ([Weis80], [GoMO06]). Thus it could be the length of the longest tour (in time) found in the network. The assumption is that the network is not fully connected. That means each node in a network needs not to have a direct connection to all other nodes. In the other words, in a fully connected network of n nodes, each node has n-1 neighbours. The update of pheromones is either "best-so-far" or "iteration-best" which depends on the fact how large instances are taken in consideration. For smaller instances, "iteration-best" is the better strategy, whereas for bigger instances, it is better to apply "best-so-far" ([DoSt05]).

As it is already stated, if "brood sorting" is used for writing information, entries are distributed on the basis of their type (similar entries stay closer to each other). Therefore, two strings that represent two URLs, are compared by using a similarity function δ ([ŠeKü09], [ŠeKü10a]). This function is based

[17] In a theoretical model explained by means of a graph, the connection between nodes is defined by arcs. In reality, arcs correspond to links (in a network).

[18] The nearest neighbour algorithm includes the following steps: 1) stand on an arbitrary node as current node, 2) find out the "lightest" arc connecting current node and an unvisited node N, 3) set current node to N, 4) mark N as visited, 5) if all the nodes in domain are visited, then terminate, else go to step 2). This algorithm quickly yields a short tour, but usually not the optimal one due to its "greedy" nature. It gives as the output the sequence of the visited nodes.

on a spatial locality that fits the best to “brood sorting” as spatial locality refers to the use of data elements within relatively close storage location.

Example:

A simple spatially basic URL comparison depends on the following rules. Namely, it is inspired by belonging to the same area (semantically) - therefore the metaphor of host is involved and “its” weigh (k_1) has a more significant impact.

If the host is identical: *add* k_1;

If the path is identical: *add* k_2**path_success_rate*;

If the host is not identical: *add* k_3** host_success_rate*, where *the success_rate* is the number of words matching in right order divided through the count of words.

The real coefficients k_1, k_2 and k_3 are configurable and can take values from [0,1] in order to normalize the similarity function. The best values for these coefficients are determined by means of fine-tuning: $k_1 = 0.6$, $k_2 = 0.4$, $k_3 = 0.2$.

For example, in case of some URLs, the obtained values are presented right:

http://www.test.org/german/docs/aaa http://www.test.org/german/docs/aaa	→Identical host and path: 0.6+0.4=**1**
http://java.sun.com/german/docs/aaa http://java.sun.com	→ Identical host: **0.6**
http://www.test.org/german/docs/aaa http://java.sun.com/german/docs/aaa	→ Identical path: **0.4**
http://www.sun.com/books/docs/plane http://java.sun.com/german/docs/aaa	→ 0.2*2/3 for host + 0.4*1/3 for path: **0.26666668**

Adapting and re-modelling of these ant algorithms for the case of location and retrieval of data comprises the following changes ([ŠeKü09], [ŠeKü10a]), implemented in *constructSolution* and *depositPheromone* procedure. As already explained, procedure *constructSolution* means that the ant made a path and possibly found the data on that path. The interest is to find the best path, but also to find the data with a good quality. As the result of an ant’s searching for the specified data, the following situations are possible to happen: *no data* found, an *exact data* found, and an *acceptable data* found with the accuracy < ε, where ε is an error rate (configurable parameter, real number) given in advance, connected to the definition of δ. Namely, the general form of the similarity function is: $\delta = \delta$(current_solution, exact_solution), that describes how good (acceptable) solution is found, $\delta \in [0,1]$. The type of the similarity function δ can be changed, however, its value are normalized (into segment

[0,1]). This implies the following changes in *depositPheromone* procedure and consequently, the ant's actions:

- *Action 1*: deposit a full amount of pheromone, if an ant found the exact data on its trip, i.e., if δ(current_solution, exact_solution) = 1;
- *Action 2*: deposits less amount of pheromone, if an ant found acceptable data on its trip with the accuracy < ε, i.e., δ(current_solution, exact_solution) < ε;
- *Action 3*: Skip depositing pheromones on the trip[19], if an ant did not find data, i.e., δ(current_solution, exact_solution) = 0;
- *Action 4*: Assign some negative values of pheromones, i.e., decrease the values of pheromones more than they are in unvisited arcs, if an ant did not find data.

These actions describe a depositing of a different amount of pheromones according to the quality of solution found. Therefore, *DepositPheromone* procedure is adapted for this problem of interest in the following way that includes the results of the numbered actions, i.e., the value of δ:

1) for MMAS Ant System algorithm:

$$\Delta\tau = \frac{1}{MC^{best}} \tag{5.12}$$

where $\Delta\tau$ is the amount of pheromone added (Eq.4.3.) and $M = \dfrac{1}{\delta}$.

2) for AntNet algorithm:

$$\tau \leftarrow r \cdot (1-\tau) \cdot \delta \tag{5.13}$$

In both cases, the "distance" matrix (heuristic distance) is calculated by means of "time": the distance between lookup container$_i$ and container$_k$ is not expressed as geographical distance; it is the amount of time needed to go from lookup container$_i$ to lookup container$_k$. Ants are positioned randomly.

Ant Algorithms for Load Balancing

As already stated in Chapter 3, a load balancing algorithm is responsible for the realization of the location policy. Ant algorithms used for load balancing are MMAS and AntNet. Actually, the first phase – construct solution- uses again Eq. 5.1. for the case of MMAS, i.e., Eq.5.6. and Eq.5.7. for the case of AntNet. The pheromone update phase is done by using Eq. 5.12. for

[19] The values on arcs (links) it traversed will be the same as the values on the rest arcs in the network.

MMAS and Eq.5.13. for AntNet, where the suitability function δ is defined in Table4.3. and its argument has the values defined in Eq. 5.18. A detailed explanation about suitability function δ in case of load balancing is presented in subsection 5.3.

5.3 Bee algorithms

5.3.1. Bees Behaviour in Nature

One bee colony in nature consists of bees with different roles defined below ([CaSn91]): foragers, followers, and receivers. A bee colony demonstrates a natural intelligence that performs self-organization through two types of behaviour: navigation and recruitment. The navigation means searching for nectar in an unknown landscape. It is non-pheromone based, and thus another strategy - so called path integration ([LMLPW00]) - is used for orientation. A forager bee is capable to "compute its present location from its past trail continuously. So, path integration is the insect knowledge of direction towards and distance from its destination. A forager scouts for a flower with good nectar, returns to the hive, unloads nectar, and performs a recruitment strategy. The recruitment means that a bee communicates the knowledge about the visited flowers to other bees ([CaSn91]), i.e., it "advertises" the visited flower site. For the recruitment, a bee uses a special strategy for direct communication with its hive mates, so-called waggle dance. Using this "dance language" on the vertical combs in the hive, bee informs its hive mates about the direction, distance and quality of the food found. The better the quality of the nectar source and the shorter the distance from the hive is, the longer a forager's dance duration ([vonF67]). A follower randomly chooses a forager whom it follows and visits the flower that has been "advertised" without own searching. A forager can choose to become a follower in the next step of navigation, and vice versa. A receiver always stays in the hive and processes the nectar.

High autonomy, distributed functioning, and self-organization characterize the biological bees behaviour ([CaSn91]). Bees solve the problem in a collective decision making process. Although these characteristics are similar to ants behaviour, the difference is presented in subsection 4.1. Bees communicate directly (non-pheromone based communication), whereas ants communicate indirectly (pheromone based communication).

Bee-inspired algorithms have been applied to several computer science problems like Travelling Salesman Problem ([WoLC08]), job shop scheduling ([CSLG06]), routing and wavelength assignment in all-optical networks

([MaTA07]), training neural networks for pattern recognition (Pham et al. 2006), scheduling jobs for a production machine ([PKLP07), computer vision and image analysis ([OlPu06]). These problems benefited of using bee intelligence. Although some of these applications treat a kind of job scheduling, it differs a lot from our approach. Namely, they used a simplified version of a scheduling problem by including several limitation given in advance (e.g., a single machine supplies jobs, each job needs only one operation to be executed, etc.).

5.3.2. Bee Algorithms for Application Scenarios

As bee algorithms have neither a general form nor the theoretical foundation, this subsection starts immediately with the bee algorithm for the application scenarios. Bee algorithms have not been used for these scenarios before, so this is a novelty as they are applied for the first time to these problems. At the end, new theoretical results about the convergence of the presented form of bee algorithm are derived and proven.

Bee Algorithm for Dynamic Load Balancing

The principals for usage of bee intelligence for load balancing are described in ([ŠeKü08], [ŠeKü10c]). The way of mapping this process from nature to a heterogeneous distributed system is abstracted due to the model that corresponds to the dynamic load balancing problem and includes the following notions. The notions from distributed systems are defined as:

- Software agents represent bees at the particular nodes.
- A node contains exactly one hive and one flower, where a flower can have many nectar units that can be taken out by a bee and a hive has a finite number[20] of receiver bees and outgoing (i.e., forager plus follower) bees.
- A task represents one nectar unit.

At the beginning of the process, all outgoing bees are foragers as the population is without any information about the environment. Foragers perform two described strategies: they navigate, i.e., scout for a location policy partner node of their node to pull or push nectar from/to it, and they recruit followers. In the rest of the text, the emphasis will be put on the outgoing bees (foragers and followers) as they are the main actors in the algorithm (i.e., they perform the strategies: navigation and recruitment). The receivers only process tasks at their node and have no influence on the algorithm. The goal is to find the best

[20] A population has finite number of individuals.

location policy partner node by taking the best path which is defined to be the shortest one. A suitability function δ defines the best location policy partner and could take any form, e.g., from chapter 5, Table 5.2.

Phase **1** – Navigation: A bee goes from one node to another until one of these situations occurs:

- A bee found the best location policy partner,
- A bee examined all nodes in a network without results, i.e., made a "full" path if the network is fully connected[21],
- A bee examined some nodes in a network without result, i.e., made a "partial" path if the network is not fully connected.

After finishing navigation, a bee goes back directly to the hive (which is different in comparison to ants behaviour; ant uses the same path to go back to the nest). This difference is explained in subsection 4.1.
A navigation strategy determines which node will be visited next. It is mathematically described and realized by a stochastic state transition rule[22] ([WoLC08]):

$$P_{ij}(t) = \frac{[\rho_{ij}(t)]^{\alpha} \cdot [\,1/d_{ij}\,]^{\beta}}{\sum_{j \in A_i(t)} [\rho_{ij}(t)]^{\alpha} \cdot [\,1/d_{ij}\,]^{\beta}} \tag{5.14}$$

where $\rho_{ij}(t)$ is the arc fitness from node i to node j at time t and d_{ij} is the heuristic distance between i and j, α is a binary variable that turns on or off the arc fitness influence and β is the parameter that controls the significance of a heuristic distance.

In the calculation of the arc fitness values, two situations are possible:

- A bee is forager

According to the state transition rule, arc fitness values are $\rho_{ij} = 1/|N_i|$, where $|N_i|$ is the number of neighbouring nodes of node i. A forager can decide to become a follower in the next cycle of navigation.

- A bee is follower

Before leaving the hive, bee observes dances performed by other bees and randomly chooses to follow one of the information offered through these dances. This information contains the set of guidance moves that describes the

[21] It is defined on page 87.

[22] This rule is a derived from the basic statistics' rules and represents the probability of choosing the path, i.e., arc (i,j) when being located at the node i, where j is one of the nodes from the neighbourhood of node i.

tour from the hive to the destination previously explored by one of its hive mates. This is the so-called *preferred path* ([WoLC08]). When a bee is in a node *i* at time *t*, two sets of next visiting nodes can be derived: the set of allowed next nodes, $A_i(t)$ and the set of favoured next node, $F_i\,(t)$. $A_i(t)$ contains the set of neighbouring nodes of node *i*, whereas $F_i\,(t)$ contains a single node which is favoured to reach from node *i* according to the preferred path. The arc fitness is defined as ([WoLC08]):

$$\rho_{ij}(t) = \begin{cases} \lambda & \textit{if } j \in F_i(t) \\ \dfrac{1-\lambda \cdot |A_i(t) \cap F_i(t)|}{|A_i(t)| - |A_i(t) \cap F_i(t)|} & \textit{if } j \notin F_i(t) \end{cases} \qquad \forall j \in A_i(t),\ 0 \le \lambda \le 1 \quad (5.15)$$

where $|S|$ denotes of the cardinality (i.e., the number of elements) of set *S* and λ is the probability of "following a node" in the preferred path. So, $|A_i(t) \cap F_i(t)|$ can be either 0 or 1, i.e., $A_i(t)$ and $F_i\,(t)$ may have either none element or only one element in their intersection.

<u>Phase **2** – Recruitment:</u> A recruitment strategy exchanges the obtained knowledge between bees about path (distance) and quality of the solution. From these, we can derive a new fitness function

$$f_i = \frac{1}{H_i}\delta \qquad (5.16)$$

for a particular bee *i*, where H_i is the number of hops on the tour, and δ is the suitability function of the solution. If bee *i* found a highly suitable location policy partner node, then its fitness function, f_i, will obtain a good value. The colony's fitness function is the average of all fitness functions (of each outgoing bee):

$$f_{colony} = \frac{1}{n}\sum_{i=1}^{n} f_i \qquad (5.17)$$

where *n* is the number of outgoing bees. After a trip, an outgoing bee determines how "good it was" by comparing its result with the average value (Eq.5.17), and based on that decides its next role (a forager or a follower) which is presented in Table5.1. ([NaTo04]). The success of a bee affects the credibility of its recruitment, expressed as a quotient between f_i and f_{colony}.

Table 5.1. Lookup table for adjustment of probability to follow ([NaTo04]).

scores	probability to follow
$f_i < 0.5*f_{colony}$	0.60
$0.5*f_{colony} \leq f_i < 0.65*f_{colony}$	0.20
$0.65*f_{colony} \leq f_i < 0.85*f_{colony}$	0.02
$0.85*f_{colony} \leq f_i$	0.00

This procedure can be described as follows ([WoLC08]):

```
procedure BCO_MetaHeuristic
   while(not_termination)
    observeWaggleDance()
    constructSolution()
    performWaggleDance()
   end while
end procedure
```

Figure 5.2. Pseudo-code of BCO metaheuristic

Both under-loaded nodes and overloaded nodes (and also concurrently) can start the location policy. The rest of the actions depend on the fact which nodes start it, so the following situations can be differentiated:

1. An under-loaded node starts the location policy: its bee searches for a suitable task belonging to some overloaded node and carries the information about how complex task the node can accept;
2. An overloaded node starts the location policy: its bee searches for an under-loaded node that can accept one or more tasks from this overloaded node. It carries the information about the complexity of tasks this overloaded node offers and compares it with the available resources of the current under-loaded node that it is just visiting.

Obviously in both situations, the complexity of the task and the available resources at a node must be compared. Therefore, the following notions are introduced ([DSCB03]):

- A host load *hl* represents the fraction of the machine that is not available to the application.
- A host speed *hs* represents the speed of the host and its value is relative in a heterogeneous environment.
- A task complexity *c* is the time necessary for a machine with $hs = 1$ to complete a task when $hl = 0$[23].

[23] Note: This is a hypothetical due to the definition from ([DSCB03]), as in reality, it can be expected that some load always exists.

On the basis of these notions, the argument x of suitability function $\delta = \delta(x)$ is calculated:

$$x = \frac{\frac{c}{hs}}{1 - hl} \tag{5.18}$$

The situation is ideal for $x = 1$ as it numerically depicts that the most adequate location policy partner will be found. The "adequacy" also comprises the issue of taking care about not to waste the available resources. For example, the situation, in which an under-loaded node with high resource capacities takes a work from an overloaded node that offers tasks with small complexity, describes a wasting of the available resources and the example of a badly matched location policy partner nodes. All mentioned parameters are configurable.

Theoretical Considerations

In the following part, basic theoretical considerations are considered:

- Does the algorithm find the optimal solution?
- Do we speak about global optimum or local optimum?
- Do we have convergence in value and/or convergence in solution?

First, we shall differentiate between ([DoSt05]):

A convergence in value: This is the evaluation of a probability that the algorithm will generate an optimal solution at least once.

A convergence in solution: The algorithm reaches the state which keeps generating the same optimal solution.

We provide a convergence in value. Generally speaking, although the convergence in solution is a stronger result than the convergence in value, in an optimization problem we are interested in finding the optimal solution once, so that the convergence in value is all that we need.

For this purpose, pre-assumptions are ([DoSt05]):

1. $G = (C, L)$, is a graph of n nodes and links (arcs) between these nodes (nodes are not necessarily fully connected in the load balancing scenario); the set of nodes is $C = \{c_1, c_2, \dots, c_n\}$, and the set of links (arcs) between nodes is $L = \{(c_i, c_j)\colon 1 \leq i, j \leq n\}$; L is associated with a distance (or cost) matrix.
2. (S, f, Φ), where S is the set of candidate solutions, f is the objective function, Φ is the set of constrains that defines the set of feasible solutions; the goal is to find an optimal solution s_{opt}; Θ is the finite set of states of the problem, $\theta = <c_i, c_j, \dots, c_h, \dots>$, $|\theta|$ is the number of nodes in a sequence, $|\theta| \leq n$; Θ^* is the set of feasible states, $\Theta^* \subseteq \Theta$;

3. for the time being, static scenarios in this theoretical explanation are considered;

The probability rule (Eq.5.14) could be described in a more abstracted way as:

$$P(c_{h+1} = j | x_h) = \frac{F_{ij}(\rho)}{\sum_{j \in A_i} F_{ij}(\rho)} \tag{5.19}$$

where F_{ij} is some non-decreasing function, $F_{ij}(z) = z^{\alpha} \eta_{ij}^{\beta}$

The next is a new result derived as the consequence of the similar result that considers convergence of one group of Ant System Algorithms in which, for example, Min Max Ant System belongs to ([DoSt05]). Therefore, the next corollary is inspired and based on one theorem from ([DoSt05]) that proves convergence in value of Min Max Ant System. The theorem says that when using a fixed positive lower bound on the pheromone trails finding the optimal solution is guaranteed for this specific group of algorithms. The next proof is based on some specifics for bee algorithms and some general issues that could be found in the proof of convergence in value of Min Max Ant System as well.

Corollary: If *P(k)* is the probability that bee algorithm finds an optimal solution at least once within the first *k* iterations, then $\lim_{k \to +\infty} P(k) = 1$.

Proof. From Eq.2 follows that the arc fitness ρ for a follower bee belongs $\rho \in \{\frac{1-\lambda}{l-1}, \lambda\}$, where λ is the probability of choosing the preferred path and l is the number of neighbouring nodes of a particular node. If the case for a forager bee is added, that means $\rho \in \{\frac{1-\lambda}{l-1}, \lambda, \frac{1}{l}\}$. So, for the given network values of arc fitness can have a finite number of values and it values stay in some closed interval [ρ_{min}, ρ_{max}]. The lower bound is positive and fixed for the given network. Therefore, any feasible choice from Eq.5.19 for any partial solution x_h is made with the probability:

$$p_{\min} \geq \frac{\rho_{\min}^{\alpha}}{(n-1)\rho_{\max}^{\alpha} + \rho_{\min}^{\alpha}} \tag{5.20}$$

Any solution (incl. the optimum solution) can be generated with the probability:

$$p > \left(\frac{\rho_{\min}^{\alpha}}{(n-1)\rho_{\max}^{\alpha} + \rho_{\min}^{\alpha}} \right)^{m} > 0 \qquad (5.21)$$

where *m* is the maximum length of a sequence. From this fact, it follows that $P(k) = 1-(1-p)^k$. For every arbitrarily small $\varepsilon > 0$, $P(k) \geq 1-\varepsilon$. That means: $\lim_{k \to +\infty} P(k) = 1$.□

The better explanation of this corollary is given through the following discussion. In Bee Algorithm, the values that are assigned to arcs are the values of arc fitness, ρ_{ij}. Some of these values will be implicitly reinforced by learning of the other hive mates via waggle dance (i.e., a recruitment process). The fact how "strong" is the recruitment of a particular bee depends on the values of suitability function δ and the path length. The higher the value of δ and the lower the path length, the stronger the recruitment is. How could convergence in value from ([DoSt05]) be transferred to bee algorithms? First, Bee Algorithm scenario forces the best-so-far solution, and uses implicit maximum value of ρ_{max} (which is directly implied by f_{max} from the best-so-far solution). Second, the value of λ is initialized to the upper limit (λ_0), so the minimum value ρ_{min} will be reached in: a) $\frac{1-\lambda_0}{l-1}$, for any case b) $\frac{1-\lambda_0}{n-1}$, for the case with fully connected nodes. Third, any feasible solution can be constructed with a nonzero probability. If we assume that connection *(i,j)* does not have the largest probability to be chosen (i.e., *j* does not belong to set F_i), then the probability of choosing this connection is $\frac{1-\lambda}{l-1}$ and this is the worst case gives in Eq.5.20.

Bee Algorithm for Information Retrieval

An intelligent overlay is constructed by using bee intelligence. Bee algorithm for Information Retrieval uses Eq. 5.14. and Eq. 5.15 from the navigation part, whereas the recruitment phase is realized by means of Eq. 5.16. and the general form of the suitability function is: $\delta = \delta$(current_solution, exact_solution), that describes how good (acceptable) solution is found, $\delta \in [0,1]$. The type of the similarity function δ can be changed, however, its value are normalized (into segment [0,1]). This function is described in 5.2.3.

5.4 Summary

Chapter 5 explained how swarm intelligence can be mapped or adapted to the located application cases. Ant intelligence algorithms can be successfully used as algorithms for an overlay network (both for writing information and for searching as two types of ant behaviour are modeled – brood sorting and food searching), and also for dynamic redistribution of load in a network. Ant intelligence is more or less known. Relatively new in IT– bee intelligence can be also successfully used as the algorithm for an intelligent overlay network. A special application of bee intelligence is in the domain of dynamic load balancing, where the second part that describes recruitment phase of bees' behavior contains an improvement and novelty. A construction of a bee algorithm for load balancing and information retrieval, and an adaptation of two ant algorithms for load balancing and information retrieval are references of this chapter. The novelty is the implementation of bee intelligence for the load balancing problem for the first time in order to improve the quality of the solution and scalability. However, transferring bio-mechanisms from nature requires the adequate mathematical models that imply a construction of the appropriate algorithms. The main common characteristic of these algorithms are: they simulate some kind of bio-intelligence, so they are intelligent algorithms; these algorithms are mostly heuristics and non-deterministic.

The main difference between bee intelligence and ant intelligence is in the way of communication: ant communicate indirectly, bees communicate directly. Software agents that play role of ants "communicate" asynchronously, whereas software agents that play role of bees "communicate" synchronously.

At the end, the convergence in value of bee algorithm is proven.

CHAPTER 6

6 BENCHMARKS AND EVALUATION

This chapter contains the simulation results, obtained in both application scenarios: information retrieval and dynamic load balancing. First, test examples are described, and then the results are presented. After that, the evaluation and analysis of the results are elaborated. In subsection 1.4., the methods used to evaluate the claims of thesis are presented. One of these methods is the usage of benchmarks. These benchmarks serve to answer research questions 4 (what is the best parameter tuning?) and 5 (is it better to have an intelligent or unintelligent approach or combination?) as well as one part of research questions 1 (can these IT real use cases profit from the usage of self-organization?) and 2 (can the principles of self-* help to cope with complexity in heterogeneous distributed systems; what could be improved by employing self-*? How and in which extent the employed principles of self-organization improve performance and scalability in the two application scenarios?). In these benchmarks, the performance of the system and scalability are measured. Therefore, the benchmarks serve to prove the improved performance and scalability of a system by employing self-* mechanisms. The performance is an important property as the tasks set in systems become much more complex over time, and consequently also the demands imposed on the systems (in terms of the complexity, number of transactions, number of users, etc). The scalability of a system is closely related to performance. However it focuses on the predictability of the system's performance as the workload increases ([RoWo05]). The necessity of a fast adaptation to new requirements and changes in the environment is highly important as even if the system meets its goals today, there are no guarantees that it will meet goals in the future, be able to cope with increased numbers of users, transactions, messages or to handle increased complexity of processing. The performance is expressed in absolute execution time ([KaBo04]). In both scenarios, the benchmarking included: swarm intelligence algorithms (ant algorithms and bee algorithm) and unintelligent algorithms.

The first part (section 6.2.) considers the results obtained in Information Retrieval scenario. The swarm intelligence algorithms (first, ant algorithms: MMAS and AntNet, and then, bee algorithm) are benchmarked and compared with Gnutella lookup mechanisms. Gnutella was chosen for a comparison as it is the most similar to the systems used in this thesis: unstructured P2P, purely decentralized. Therefore, Gnutella is a well-known representative from this group. The benchmarking is done in two test environments (see subsection 6.1.2.). Tests on a cluster (6.1.3.) are done first to identify the best set of parameters and to obtain the "preliminary results". Later, tests are done on the Amazon Cloud (6.1.4.).

The second part contains the results of Dynamic Load Balancing. This part included benchmarking the basic SILBA (6.2.1). First, the swarm intelligence algorithms (ant algorithms: MMAS and AntNet, and bee algorithm) are benchmarked and compared with Random algorithm, Sender algorithm and adapted genetic algorithm. The intention was to perform the benchmarks by comparing swarm based intelligence algorithms with unintelligent algorithm and some other intelligent (non-swarm) algorithm. Genetic algorithms are famous metaheuristics. Unintelligent algorithms (Random, Sender) are chosen due to the fact that they represent the base algorithms from the conventional approaches. Sender initiated algorithm refers to the triggering from over-loaded nodes. Hence, this algorithm is chosen for the comparison with swarm based algorithms as they obtained the best results when triggering from overloaded (and OK) nodes (although, they allow for symmetric triggering). Random algorithm is neutral. Later, the benchmarking is done for the extended SILBA (6.2.2.). As the extended SILBA allows for dynamic load balancing on different levels concurrently (e.g., between subnets in a network and inside subnets), different algorithms are plugged into different levels and their combinations are investigated. This approach is taken in consideration as the intention was to detect which combination of algorithms fits the best to a particular network topology (chain, ring, full, star); also, detection which topologies profit the most from the application of swarm intelligence is investigated. The numbered topologies are chosen according to the fact that they fit the best to the description of patterns (and pattern composition) from chapter 5.2. Namely, these topologies were applied to subnets and it is assumed that subnets might have one or more nodes in the intersection.Tests on a cluster (6.2.1.) are done first to identify the best set of parameters and to obtain the "preliminary results". Later, tests are done on the Amazon Cloud (6.2.2.).

The number of sampling of nondeterministic algorithms was chosen according to the fact how quickly the algorithm converges, and whether the obtained results are uniform (without peaks). As a system was stable all the time, it implied the conclusion that the results are reliable. From the other side, scalability of a system is also proven in these benchmarks. Hence, the further increasing the dimensions of instances (i.e., the proportional increasing both the load and resources) would not jeopardize the stability of a system. The maximal dimensions of benchmarks are implied by a waiting time (huge dimensions responded to "too long" waiting times).

6.1 Results (Information Retrieval Scenario)

Information Retrieval benchmarks start with the explanation of test examples and test environments in order to present raw results obtained in different test environments.

6.1.1 Test Examples

Test examples are constructed on the following way.

1) For ant intelligence:

Two algorithms of writing data into containers and two algorithms of performing lookup and retrieving data from containers are implemented. Each of their combination is performed (Table 6.1). Namely, the intention was to detect the best combination, while each combination refers to one algorithm of writing data into container plus one algorithm of performing lookup (e.g., random writing plus MMAS for lookup is one combination denoted in Table 6.1. as 1st case).

Table 6.1. Possible combinations used in benchmarks

	MMAS	*AntNet*
random	1st case	2nd case
brood sorting	3rd case	4th case

2) For bee intelligence: as "brood sorting" is the mechanism of ant colony in nature, random writing of data into container is done and combined with bee algorithm for lookup.

As the used algorithms are non-deterministic, all test examples were evaluated 10 times (enough number of sampling for one nondeterministic algorithm in order to conclude whether it gives consistent results or it deviates and gives suboptimal solution) and the average values were found. The benchmarks are grouped into two groups:

- first, the behaviour of the implemented algorithms is analyzed by means of different combinations and different parameters' settings, as well as the obtained scalability; for each of the numbered combinations, fine-tuning of parameters is done in order to discover which parameters' settings fit the best to which combination; afterwards, all "winners" are compared;

- second, the query capability of the system is investigated and compared to Gnutella lookup mechanism.

6.1.2 Test Environments

Two different test environments were used: a cluster of 4 machines, and the Amazon EC2 Cloud. Each machine of the cluster had the following characteristics: 2*Quad AMD 2,0GHz with 16 GB RAM. We simulated a network with 16 (virtual) nodes. Each test run begins with a “cold start" and all nodes being UL. On Amazon Cloud [ACloud11], we used standard instances of 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of local instance storage, and the 32-bit platform. As already stated, tests are done first on a cluster to identify the best set of parameters and to obtain the “preliminary results”. Later, tests are done on the Amazon Cloud as the real environment.

6.1.3 Results obtained on the Cluster

In this part, first the fine-tuning of parameters is presented with the results. Later, query capability of a system is measured and compared to Gnutella lookup mechanism. Besides, load scalability is investigated.

Fine tuning of parameters

The performances of different ant algorithms are compared (Table 6.1). A great amount of work in the benchmarks is dedicated to the fine tuning of parameters in order to find their best possible combination for the solution. The setting of parameters is chosen according to their predefined range ([DoSt05]):

- *For MMAS* – α varied from 0 to 1 with the step of 0.5; β varied from 2 to 5 with the step of 1.0; ρ varied from 0.5 to 0.9 with the step of 0.2.
- *For AntNet* – α varied from 0.2 to 0.5 with the step of 0.1; C_2 varied from 0.15 to 0.35 with the step of 0.05.

The rest of parameters for AntNet were based on the following values ([DoSt05]): 0.005 for the used exponential mean coefficient, 0.3 sec for the time interval between two generations, 15 for the maximum length of ant's life (in hops), 0.3 for the maximum length of the observation window and 0.7 for the value of C_1. The number of ants was 10.

Except measurement of the absolute execution time, the issue of investigation was scalability, i.e., different types of scalability. In this first group of benchmarks for the interpretation of scalability, we focused on the *space scalability.*

"Space scalability. A system or application is regarded as having space scalability if its memory requirements do not grow to intolerable levels as the number of items it supports increases. Of course, *intolerable* is a relative term. We might say that a particular application or data structure is space scalable if its memory requirements increase at most sublinearly with the number of items in question" ([Bond00]).

The results in this section, represented graphically, reflect the performance, i.e., one of the performance measures – time. The memory requirements consider the container's size, while the number of lookup containers was increased. The benchmarks were performed with the memory requirement (i.e., the container size) of 13000B, 26000B and 39000B. However, increasing the container size did not influence the performance. At the beginning, the best possible combination of parameters is analyzed in all considered cases. Figure 6.1 describes the best-obtained results while varying of parameters in the first case (the combination of random positioning of data in the network and the lookup mechanism based on MMAS algorithm while treating only one query). The following cases are compared: the best obtained combination in the situation when the value of α was positioned on 0.0 and the rest of parameters were varying (blue line), the best obtained combination in the situation when the value of α was on the next step (0.5) and the rest of parameters were varying (pink line) and the best obtained combination in the situation when the value of α was 1.0 and the rest of parameters were varying (yellow line). Obviously, the best results are obtained for $\alpha = 0.0$, $\beta=5.0$, $\rho=0.5$. Because of that, we graphically presented this case (Figure 6.2). In order to illustrate the fine-tuning of parameters, one part of benchmarks is presented in the Appendix C.

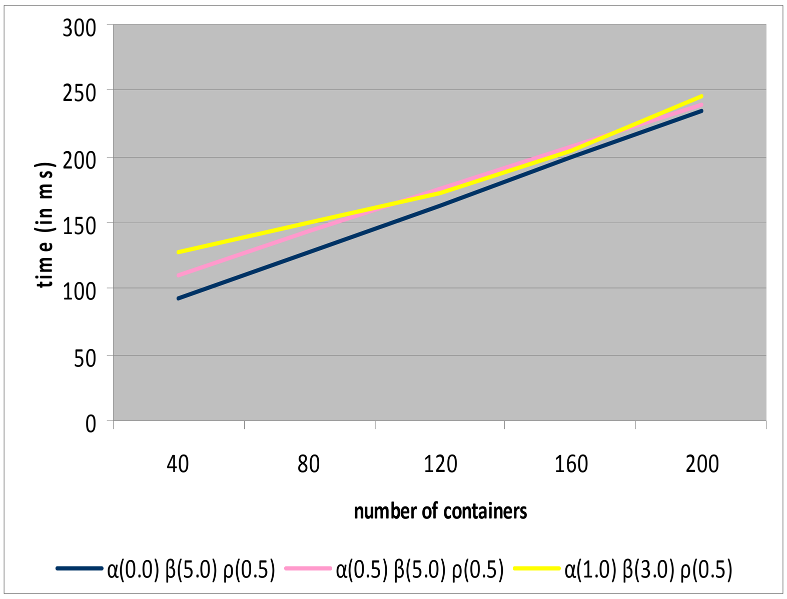


Figure 6.1. Different kind of combination for first case (Random/ MMAS) ([ŠeKü10a]).

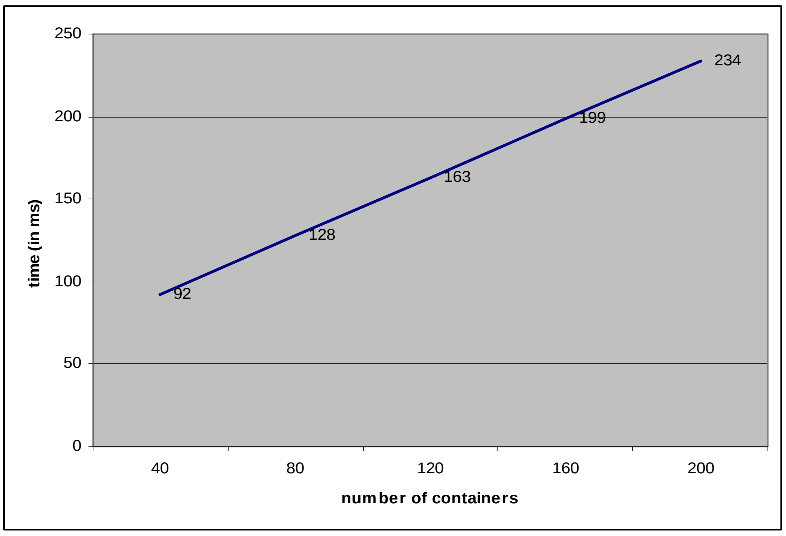


Figure 6.2. The best combination (1st case): Random/ MMAS (α= 0.0, β=5.0, ρ=0.5) ([ŠeKü10a])

The next benchmarks are based on the 2nd case for Table 6.1, i.e., contain the combination of random positioning of data in the network and the lookup mechanism based on AntNet algorithm while treating only one query. These benchmarks were also performed by tuning of parameters and investigating the different combinations. Figure 6.3 represents the best results obtained in this case with the following setting according to their predefined range ([DoSt05]):

$$\alpha = 0.2, \quad C_2 = \begin{cases} 0.35, \text{ for number of containers} = 40 \\ 0.25, \text{ for number of containers} > 40 \end{cases}$$

Further, the best results obtained in the first and the second case are compared (Figure 6.4). AntNet algorithm shows better results than MMAS. The possible reason for that is: as AntNet algorithm itself is more suitable for dynamic scenarios, it supports better the dynamic behaviour in our system while treating one query.

Although the complete benchmarking in all combinations include further the fine-tuning of parameters, for the next cases we give only the best obtained results with the designated set of parameters.

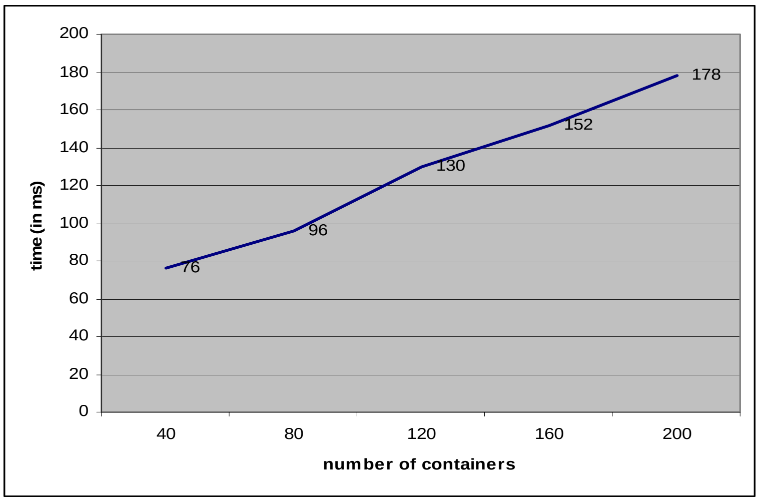


Figure 6.3. Second case: Random/ AntNet ([ŠeKü10a]).

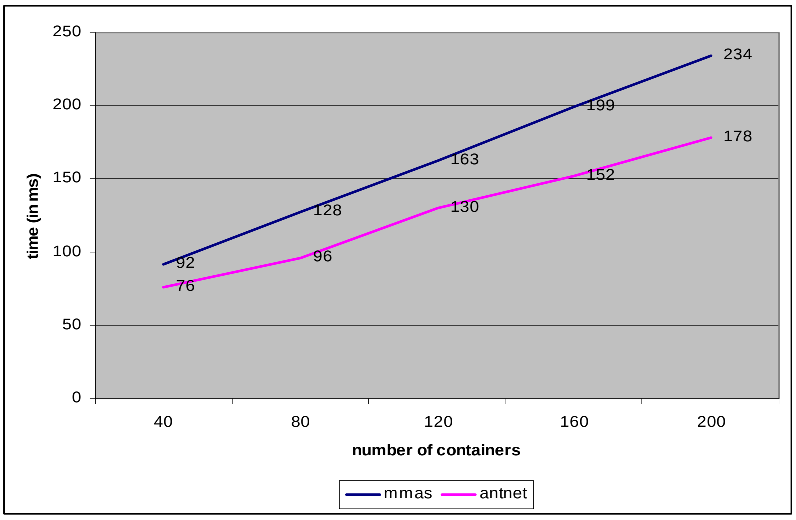


Figure 6.4. The comparison between the best obtained results in the 1st and the 2nd case (Random/MMAS vs. Random/AntNet) ([ŠeKü10a]).

Figures 6.5 and 6.6 show the comparison between cases 1 and 3, and cases 2 and 4 respectively. According to these results, Random/MMAS give better results compared with Brood/MMAS. A similar situation can be seen also on the Figure 6.6.

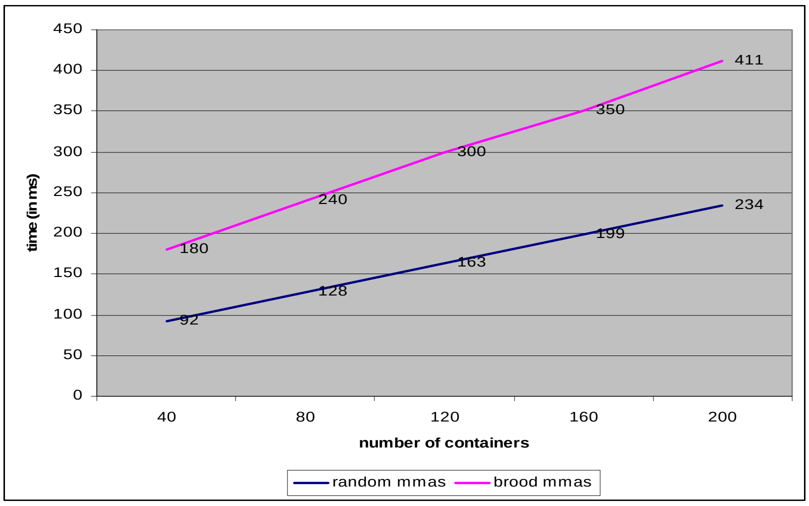


Figure 6.5. The comparison between the best obtained results in the 1st and the 3rd case (Random/MMAS vs. Brood/MMAS) ([ŠeKü10a]).

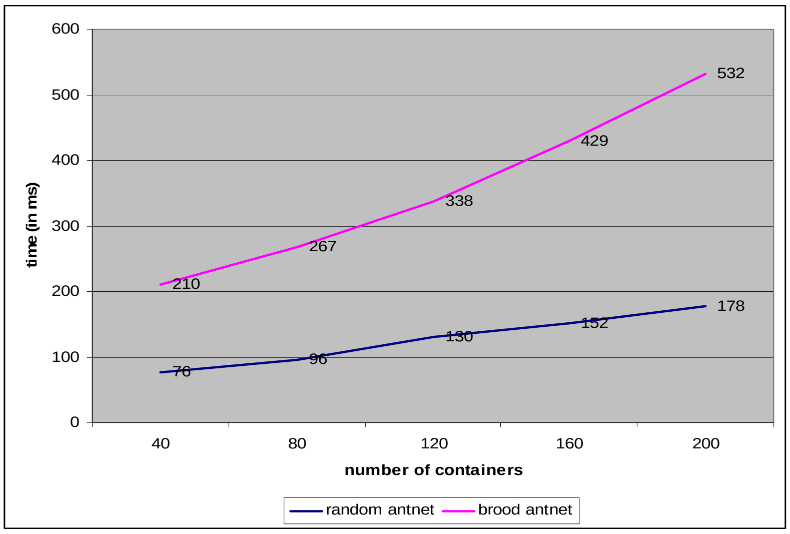


Figure 6.6. The comparison between the best obtained results in the 2nd and the 4th case (Random/AntNet vs. Brood/AntNet) ([ŠeKü10a])

This first group of benchmarks distinguishes between different ant algorithms choosing the best one with the adequate parameters' settings for the given type of problem. When comparing two different ways of lookup, the second one based on AntNet algorithm supplies a better performance (in case of retrieving only one query). Additionally, as the increasing of the container size does not influence the performance, the support of swarm intelligence provides space scalability. Therefore, the usage of XVSM enriched by swarm intelligence provides many benefits to this coordination model.

Raw results on different lookup mechanisms

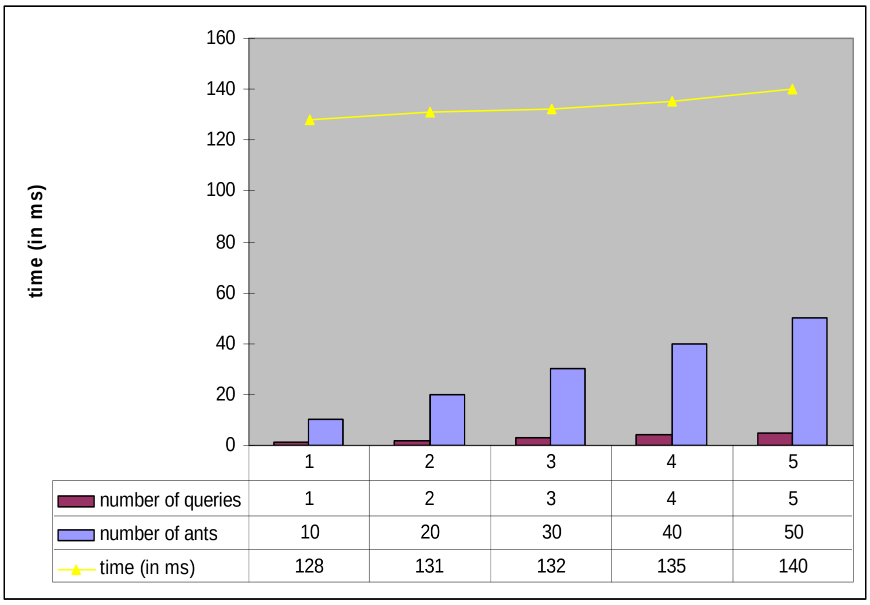
The query capability of the system is measured and the presented lookup mechanism is compared to Gnutella lookup mechanism. Second group of benchmarks focused on the issue of *load scalability* ([VaVS98]). The notion of load scalability is simplified to the version of interest to the discussed problem and quantitatively described. This restricted aspect of scalability is expressed on the basis of three dimensions: computational resources available (*R*), load of the system (*L*) and performance (*P*). Load scalability can be quantified by means of a "scalability ratio" r_{scal} for a given constant *k*

$$r_{scal} = \frac{P(L,R)}{P(k \cdot L, k \cdot R)} \tag{6.1}$$

Usually, P is a function of L and R. A constant remaining value of P when simultaneously increasing L and R by the same factor leads to the “ideal” scalability ratio of 1.

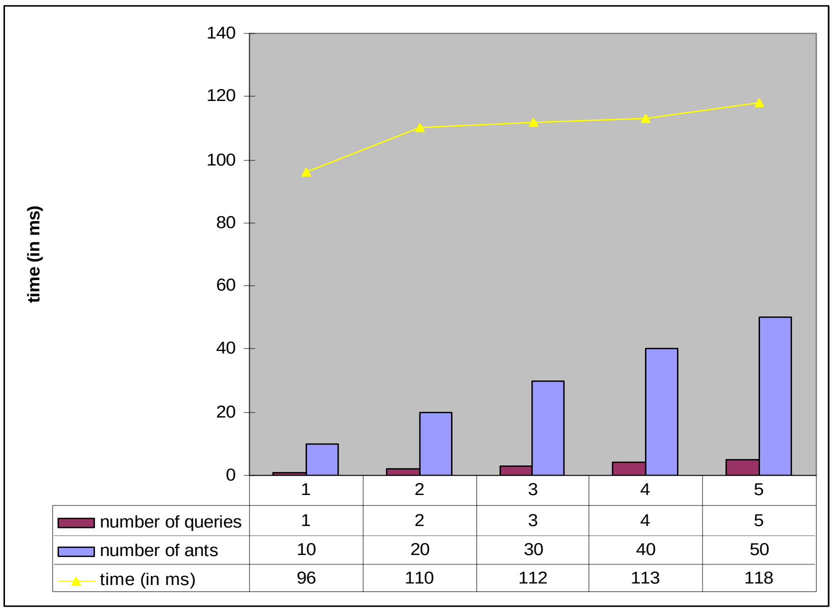
The second group of benchmarks considers an increasing of the load expressed by the number of queries needed to lookup. The number of ants is proportionally increased in order to preserve the meaning of the ant population. For example, if the number of ants is 10 and the number of queries is 5, then the real ant’s behaviour and some possible convergence of process is under the sign of question (we may assume that only 2 ants might try to find one query). Figures 6.7 and 6.8 depict the performance (measured in milliseconds) of the lookup mechanism performed by MMAS and AntNet respectively. These benchmarks are done on the network with 80 containers with the container size equals to 26000B (this number was chosen as increasing the container size did not influence the performance).

According to these preliminary benchmarks results, the presented algorithms cope successfully with the increasing number of queries, compounded of several simple queries. The possibility of increasing the number of ants that could work concurrently is used. Further, the presented intelligent lookup mechanism is compared with Gnutella lookup (Table 6.2), that is implemented by using the description from [AnSp04]. The results presented in Table 6.2. for Gnutella reflect the time when the first query is found. In order to retrieve the complete information by using Gnutella lookup, considerably much time would be needed.



	1	2	3	4	5
number of queries	1	2	3	4	5
number of ants	10	20	30	40	50
time (in ms)	128	131	132	135	140

Figure 6.7. The lookup mechanism performed by MMAS with different number of queries ([ŠeKü10a]).



	1	2	3	4	5
number of queries	1	2	3	4	5
number of ants	10	20	30	40	50
time (in ms)	96	110	112	113	118

Figure 6.8. The lookup mechanism performed by AntNet with different number of queries ([ŠeKü10a]).

Table 6.2. A comparison of the performances of different lookup mechanisms ([ŠeKü10a])

Load	Resources		Performance	
number of queries	number of nodes	number of ants	algorithm used	time (ms)
1	80	10	MMAS	128
		10	AntNet	96
			Gnutella	845
2	80	20	MMAS	131
		20	AntNet	110
			Gnutella	1216
3	80	30	MMAS	132
		30	AntNet	112
			Gnutella	1824
4	80	40	MMAS	135
		40	AntNet	113
			Gnutella	2937
5	80	50	MMAS	140
		50	AntNet	118
			Gnutella	4935
1	120	10	MMAS	163
		10	AntNet	130
			Gnutella	1140
2	120	20	MMAS	176
		20	AntNet	168
			Gnutella	1635
3	120	30	MMAS	185
		30	AntNet	177
			Gnutella	2666
4	120	40	MMAS	201
		40	AntNet	194
			Gnutella	4292
5	120	50	MMAS	220
		50	AntNet	215
			Gnutella	7211

.

From Table 6.2, it can be seen that the presented intelligent approach outperforms[24] the Gnutella lookup by means of the obtained performance. Additionally, taking into account increasing of load and increasing of the resources, according to formula (6.3), the obtained scalability is satisfactory ([JoWo00]). Finally, a graphical representation of one case (80 containers) is given in Figure 6.9.

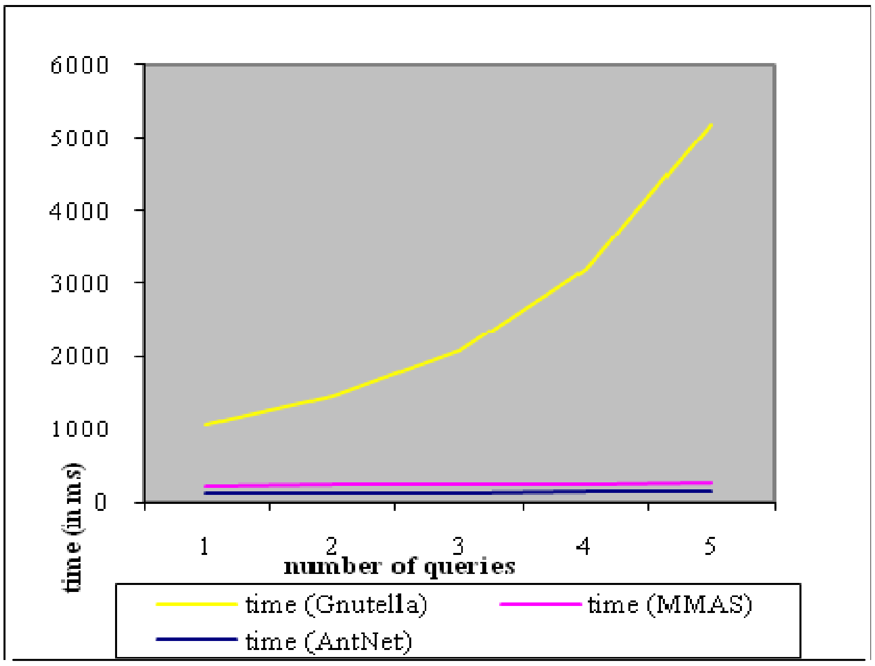


Figure 6.9. Comparison of performances of different lookup mechanisms on 80 containers ([**ŠeKü10a]**).

The benchmarks from this last subsection show that the presented system supports a larger number of queries, navigates successfully through the network of data and scales well. Note: although 4 physical machines were used, the number of lookup containers as virtual nodes in our overlay network was up to 200; so the scalability was investigated on a larger number of nodes. The benchmarks presents the definition and implementation of a new overlay network with an intelligent lookup mechanism based on swarm intelligence that is able to navigate successfully through the network of data and that scales well ([ŠeKü09], [ŠeKü10a]).

[24] The rationale is provided at the end of section 6.1.

One of the performance measures was ***the quality of found data***. In all cases, the highest data quality is obtained (data quality = 1.0 according to function δ). Because of that, this part of results is not graphically represented. The presented results, obtained on the cluster, had two-fold purpose: first, for examination of the system in one of two environments, and second, for location of the best sets of parameters (for ant algorithms) that can be applied to the other, different environments. Therefore, the next benchmarks, performed on Amazon Cloud, use the achievements obtained from these benchmarks and additionally, they are enriched by adding one more intelligent algorithm – bee algorithm.

6.1.4 Results obtained on Amazon Cloud

The benchmarks presented in this section are based on data (parameters' settings, memory requirements, etc.) **from section** 6.1.3. Therefore, the complete setting will not be repeated. Only the parameter-set identified as the best is used here and will be repeated in the description.

Fine-tuning of parameters

As it is described in 6.1.3, this case describes the results obtained by using 5 swarm intelligence algorithms and their comparison (Fig 6.10), while increasing the number of containers. The used container size is 26000B (an average size from 6.1.3.), and the parameters used are:

- For *MMAS*: $\alpha = 0.0$, $\beta = 5.0$, $\rho = 0.5$
- For *AntNet*: $\alpha = 0.2$, $C_2 = 0.35$, for number of containers = 40, i.e., $C_2 = 0.25$, for number of containers > 40
- For *Bee Algorithm*: $\alpha = 1.0$, $\beta = 10.0$, $\lambda = 0.99$

The used parameters are selected according to the best results obtained in 6.1.3., and the parameters for Bee Algorithms followed ([WoLC08]).

The rest of parameters for AntNet were based on the values [DoSt05], as explained in 6.1.3. The number of swarms was 10, and only one query was treated.

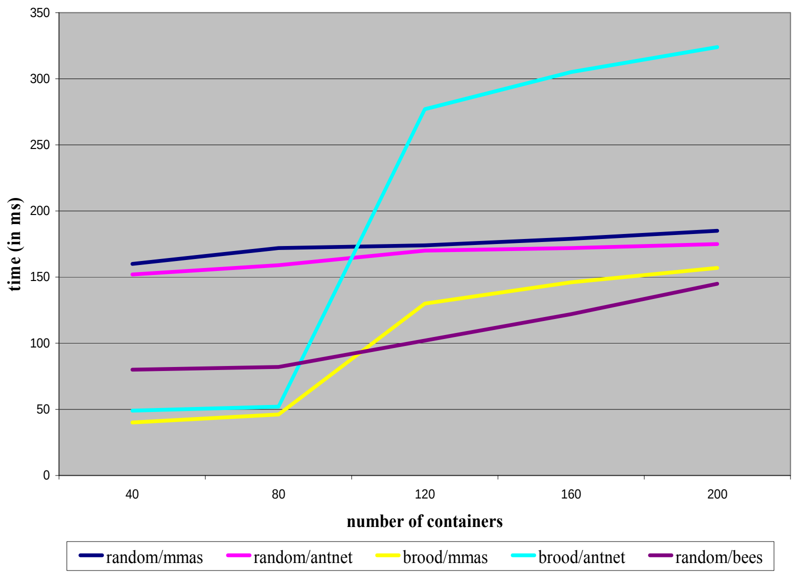


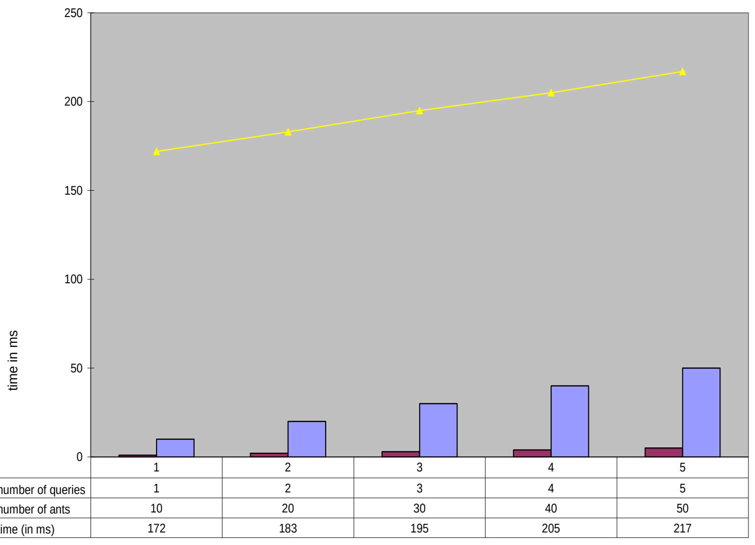
Figure 6.10. The comparison between the results of all algorithms (number of query = 1, number of swarms = 10)

In this environment, the algorithms based on a special writing technique (brood sorting) were successful with small instances. Increasing the dimensions, brood/Antnet did not obtain good results (possibly an over-clustering affected a system's robustness and did not fit to the AntNet dynamics), whereas brood/MMAS preserved the obtaining of good results. So, the behaviour of brood/MMAS is the main difference in results between these two environments. A newly introduced algorithm based on bee intelligence, obtained relatively good results on small instances (although not so good as brood based algorithms), but the best results with increasing the dimension (with bigger instances).

Raw results on different lookup mechanisms

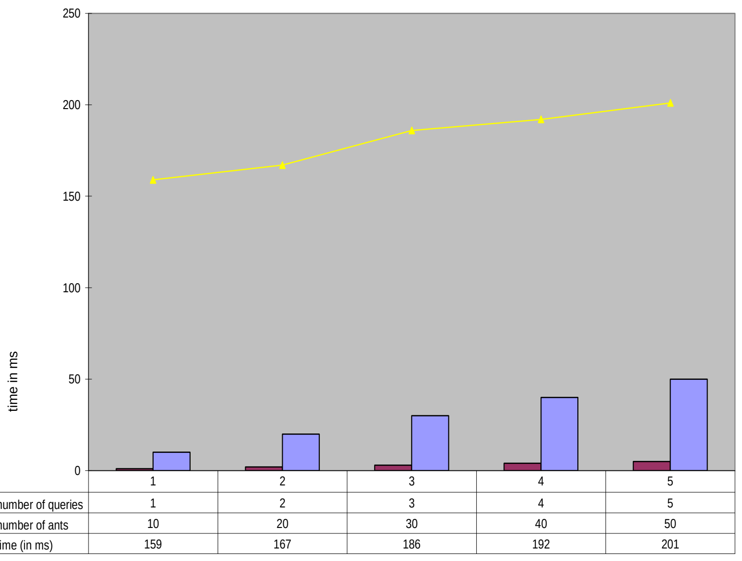
The query capability of the system is measured and the different intelligent lookup mechanism is compared to Gnutella lookup mechanism. The used parameters are the same as in 6.1.3. The next figures (6.11 - 6.15) present the results obtained by using different intelligent lookup mechanisms, while increasing the number of queries and the number of swarms. These benchmarks, presented on the figures, are done on the network with

80 containers with the container size = 26000B (this number is chosen as increasing the container size did not influence the performance).



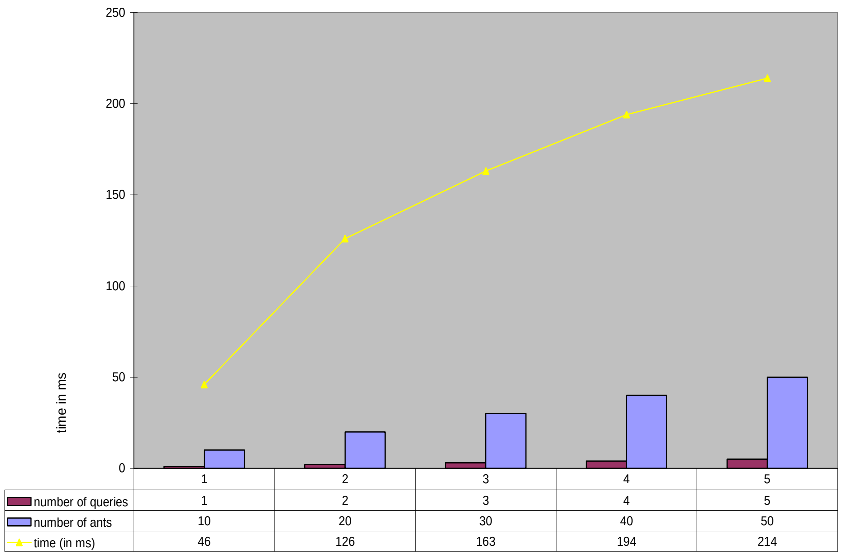
	1	2	3	4	5
🟥 number of queries	1	2	3	4	5
🟦 number of ants	10	20	30	40	50
▲ time (in ms)	172	183	195	205	217

Figure 6.11. The lookup mechanism performed by Random/MMAS with different number of queries.



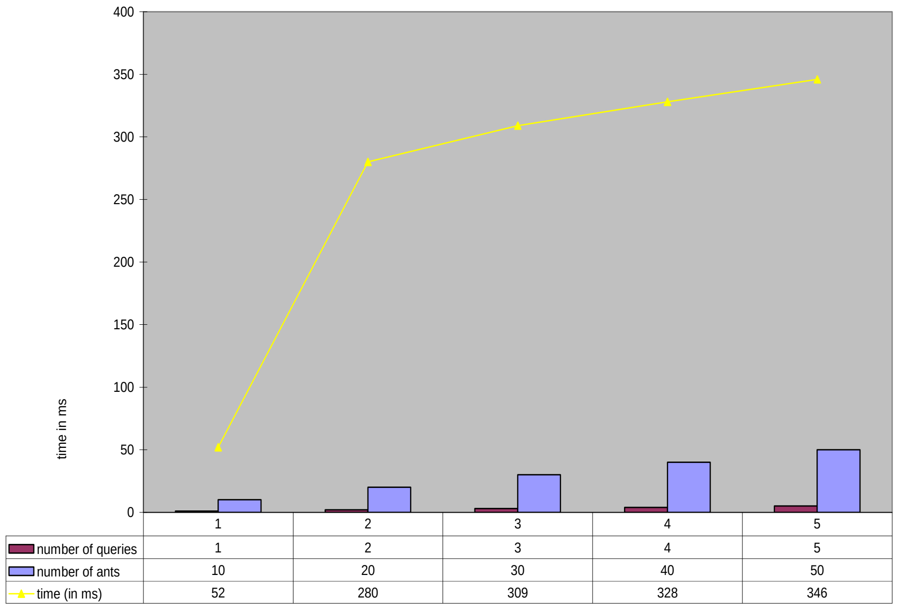
	1	2	3	4	5
🟥 number of queries	1	2	3	4	5
🟦 number of ants	10	20	30	40	50
▲ time (in ms)	159	167	186	192	201

Figure 6.12. The lookup mechanism performed by Random/AntNet with different number of queries



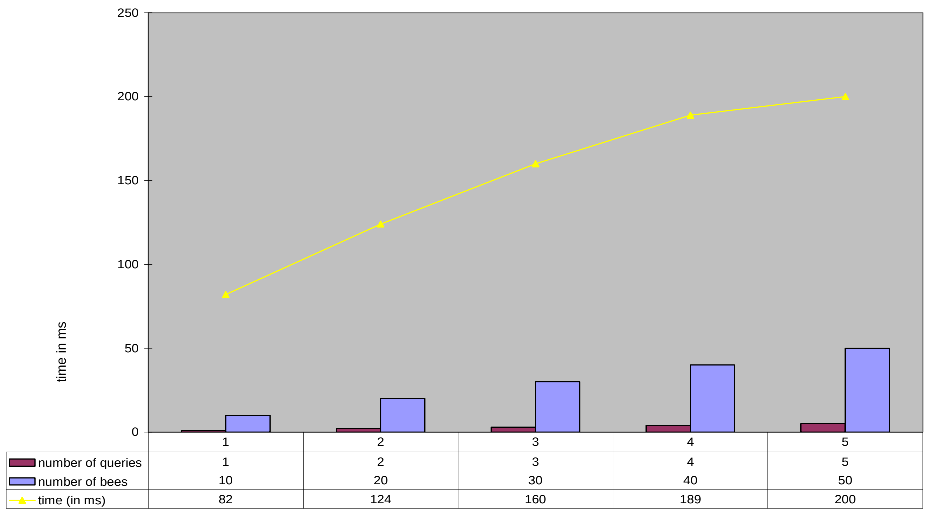
	1	2	3	4	5
number of queries	1	2	3	4	5
number of ants	10	20	30	40	50
time (in ms)	46	126	163	194	214

Figure 6.13. The lookup mechanism performed by Brood/MMAS with different number of queries.



	1	2	3	4	5
number of queries	1	2	3	4	5
number of ants	10	20	30	40	50
time (in ms)	52	280	309	328	346

Figure 6.14 The lookup mechanism performed by Brood/AntNet with different number of queries.



	1	2	3	4	5
▬ number of queries	1	2	3	4	5
▭ number of bees	10	20	30	40	50
▲ time (in ms)	82	124	160	189	200

Figure 6.15. The lookup mechanism performed by Random/Bees with different number of queries.

Table 6.3 A comparison of the performances of different lookup mechanisms (number of containers = 80)

Load	Resources		Performance	
number of queries	number of nodes	number of swarms	algorithm used	time (ms)
		10	random/mmas	172
		10	random/antnet	159
		10	brood/mmas	46
		10	brood/antnet	52
		10	random/bees	82
1	80		Gnutella	474
		20	random/mmas	183
		20	random/antnet	167
		20	brood/mmas	126
		20	brood/antnet	280
		20	random/bees	124
2	80		Gnutella	506
		30	random/mmas	195
		30	random/antnet	186
		30	brood/mmas	163
		30	brood/antnet	309
		30	random/bees	160
3	80		Gnutella	476
		40	random/mmas	205
		40	random/antnet	192
		40	brood/mmas	194
		40	brood/antnet	328
		40	random/bees	189
4	80		Gnutella	502
		50	random/mmas	217
		50	random/antnet	201
		50	brood/mmas	214
		50	brood/antnet	346
		50	random/bees	200
5	80		Gnutella	517

Table 6.4. A comparison of the performances of different lookup mechanisms (number of containers = 120)

Load	**Resources**		**Performance**	
number of queries	number of nodes	number of swarms	algorithm used	time (ms)
		10	random/mmas	174
		10	random/antnet	170
		10	brood/mmas	130
		10	brood/antnet	277
		10	random/bees	102
1	120		Gnutella	539
		20	random/mmas	194
		20	random/antnet	182
		20	brood/mmas	668
		20	brood/antnet	361
		20	random/bees	127
2	120		Gnutella	595
		30	random/mmas	199
		30	random/antnet	203
		30	brood/mmas	705
		30	brood/antnet	379
		30	random/bees	144
3	120		Gnutella	559
		40	random/mmas	210
		40	random/antnet	213
		40	brood/mmas	726
		40	brood/antnet	402
		40	random/bees	168
4	120		Gnutella	607
		50	random/mmas	222
		50	random/antnet	234
		50	brood/mmas	739
		50	brood/antnet	411
		50	random/bees	185
5	120		Gnutella	611

The previous results show the following:

- The results confirmed those ones, obtained on the cluster, i.e., among ant algorithms in this scenario, random strategy seems to be better than brood strategy. The brood based ant algorithms obtained better results than the random based ant algorithms on small instances. However, the results of the random based ant algorithms are better with increasing the dimensions.
- Another conclusion from 6.1.3. is confirmed: Random/Antnet algorithm is better than Random/MMAS; the possible reason for that could be the fact that Random/Antnet better supports dynamic processes.
- Bee algorithm obtained the best results especially on large instances; this algorithm differs from ant algorithm especialy as it informs the "starting place" of the search directly in a P2P way and, therefore got the better results.
- Scalability and preformance of each intelligent algorithm are good and outperform Gnutella lookup. Namely, the original Gnutella architecture uses a flooding (or broadcast) mechanism that supports an exhaustive optimization. From the other side, intelligent algorithms focus on important areas of the solutions space. They quickly and efficiently narrow the number of combinations to be calculated by focusing on the areas that are most profitable and most stable. Also, their advantage is that they try to find a global optimum.

6.2 Results (Dynamic Load Balancing)

This part is divided into the results of basic SILBA benchmarks and the results of extended SILBA benchmarks.

6.2.1 Basic SILBA Benchmarks

The following criterions guided the construction of the tests for the basic SILBA:

- As it is already mentioned, the balance between exploration and exploitation in each of the constructed and implemented heuristics are crucial. Hence, for each intelligent algorithm, the best combination of parameter settings – fine tuning – was the imperative.
- In order to show/prove the benefits of the SIBA framework and intelligent algorithms, compare these optimally tuned swarm based algorithms with several well-known algorithms: Round Robin, Sender, Adapted Genetic Algorithm (GA).

These benchmarks demonstrate and prove two issues. First, the agility of the SILBA pattern based framework is proved by showing that algorithms can be easily exchanged. Second, the intelligent approach based on bee algorithms showed the promising results. As some algorithms are non-deterministic (swarm algorithms, GAs), the test-examples are performed 10 times and the average values are computed[25].

Test Examples and Test Environments

The test-examples are constructed by following Table 6.5, while setting and tuning the parameters is obtained taking in consideration the objective: it was not only to measure and compare the quality of the obtained results (expressed in one of the possible metrics – time in millisec.), but also to investigate the scalability (specifically, load scalability). Therefore, the main representatives for the resources (i.e., available *nodes*) and load (*tasks*, i.e., jobs to be done) are doubled. The number of used *agents* depends on the number of nodes and the setting follows the recommended values from ([DoSt05], [DiDo98a], [ŠeKü08]). All *search modes*, forms of *suitability function* and forms of *fitness functions* are used (chapter 4). The *rest of parameters* specific for the respective algorithm, that contribute in

[25] It is explained at the beginning of section 6.1.

the equations and in formulas for the procedures Construct Solution and Deposit Pheromone took values from ([DoSt05], [ŠeKü08]), i.e., the setting follows the recommended values from ([DoSt05], [DiDo98a], [ŠeKü08]). The *constant parameters* are the initial threshold values used in the realization of the transfer policy: $T_1 = 2$ and $T_2 = 4$. However, these values will be dynamically recalculated in the process ([KüŠe09]). All combinations were performed. The load is supplied by one *client* with an arbitrary position in the network.

Table 6.5. Parameters

	bee algorithm parameters	MMAS parameters	AntNet parameters
number of nodes	4,8,16,32,64	4,8,16,32,64	4,8,16,32,64
number of tasks	50,100,200,400,800	50,100,200,400,800	50,100,200,400,800
search modes	1,2,3,4,5,6	1,2,3,4,5,6	1,2,3,4,5,6
number of bees	4,8,16,32,64		
number of ants		4,8,16,32,64	4,8,16,32,64
number of working agents	1,2,4	1,2,4	1,2,4
suitability function	0,1,2,3	0,1,2,3	0,1,2,3
fitness function	0,1,2,3	0,1,2,3	0,1,2,3
α	0,1	0.5, 1	0.2, 0.3, 0.45
β	from 8 to 12 with step 2	from 2 to 5 with step 1	
λ	0.99		
ρ		from 0.2 to 0.9 with step 0.2	
c2			0.15,0.25,0.3, 0.35

For performing test examples, the arbitrary topology is used (Fig.6.16) in which full connection between nodes were not required.

An additional investigation is done according to: different *size* of tasks and different *frequencies* of supplying tasks.

The size of tasks was also one parameter of interest. We investigated for which task size the SILBA showed the best results. The tasks are divided into three categories[26]: small (10kb), middle (20kb) and big (40kb).

[26] It was initially started with tasks of 10kB, and later the size was doubled.

Also, we investigated for which frequency of supplying tasks into the system, the SILBA showed the best results. Therefore, the frequency of supplying tasks is implemented on different ways. The idea behind is to detect whether it is better when all tasks are supplied at once or when tasks are supplied "in waves". One strategy was: everything supplied at once (at the beginning). The other strategies were: tasks are supplied into 2 waves; the first wave starts immediately and the second after a certain period of time (10 sec, 20 sec).

The same test environments are used as described in 6.1.2.

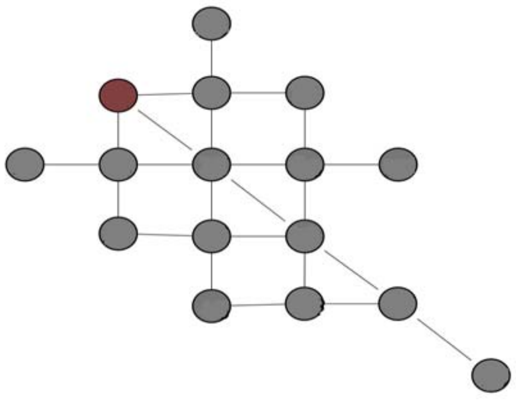


Figure 6.16. An arbitrary topology of 16 nodes: a brown node represents a client, grey nodes are the rest of nodes in the network

Raw Result Data

After benchmarking all possible combinations of parameters form Table 6.5. and investigation of the best parameters settings for each algorithm, in both environments, the best results are obtained by using the following set of parameters:

- bee algorithm: $\alpha = 1$, $\beta = 10$, $\lambda = 0.99$, suitability function 3, fitness function 2; the best search mode was 4.
- MMAS algorithm: $\alpha = 1$, $\beta = 5$, $\rho = 0.7$, suitability function 3, fitness function 2; the best search mode was 6.

- AntNet algorithm: $\alpha = 0.3$, $c_2 = 0.3$, suitability function 3, fitness function 2; the best search mode was 3.

The load is generated by one client and therefore, the system was light to moderate loaded. Under these conditions, bee algorithm showed the best results when it is triggered from OL nodes, while ant algorithms showed the best results when triggered from OK nodes (and consequently from OL nodes). The described swarm algorithms are compared with Round Robin, Sender, and Adapted GA ([ZoTe01]) that are implemented (i.e., plugged to SILBA) in the following way (routing agents of SILBA perform the algorithms):

- Random/Round Robin Algorithm: During the initialization phase, the neighbours of the current node are stored in the routing space. One of the neighbours is chosen randomly and the task is scheduled at that node.
- Sender Algorithm: The OL node triggers the routing. It is achieved by configuring OUT allocation agent (in SILBA) that is responsible for reading routing information from the space and pushing a work to another node in a network.[27]
- Adapted Genetic Algorithm (GA): The Genetic algorithm, proposed in ([ZoTe01]), is adapted in order to avoid the central coordinator. Several GAs are concurrently performed on different nodes. The size of sliding window(s) is the same for all GAs and fixed to the number of nodes, where every node has the current (waiting) task as its candidate in that window. The sliding window is not directly dependable on a client who can put tasks somewhere in the network. Each GA will reach some combination (due to the fitness function). All combinations are compared and the best one is chosen (at time *t*). This is also done by the routing agents that communicate and exchange this information. GA fires continuously until all requests are done.

The obtained results are given in Tables 6.6 and 6.7. The presented results correspond to the best search mode for the respective algorithm and they present time in ms**:**

[27] Similarly, in case that Receiver algorithm needs to be applied, where the UL node initiates the routing, another type of agent - IN allocation agent - is responsible for reading routing information from the space and pulling work from another node in a network. The Symmetric algorithm can be mapped by combining Sender and Receiver configurations.

Table 6.6. Comparison on the Cluster ([ŠeKü10c])

number of nodes	number of tasks	bee algo-rithm	MMAS algorithm	AntNet algorithm	adapted GA	sender	round robin
4	50	316833	205000	323800	592300	339880	347410
8	100	947322	542000	1037000	1896700	1391544	1142230
16	200	2889373	2720000	3685000	5311000	5366176	5711150
32	400	9823870	14142000	10334000	13278000	12646620	25423430
64	800	31927577	56217000	39534000	41832000	45586480	87865100

Table 6.7. Comparison on the Cloud ([ŠeKü10c])

number of nodes	number of tasks	bee algo-rithm	MMAS algorithm	AntNet algorithm	adapted GA	sender	round robin
4	50	11782	15000	16000	26000	16553	16748
8	100	28396	31000	33000	45000	35996	32408
16	200	60556	61000	62000	63000	63248	64817
32	400	122703	125000	128000	135000	156638	195319
64	800	257385	273000	288000	298000	413426	807118

Graphical representation and comparison of the results from the above presented are shown on Fig.6.17 and Fig.6.18[28] (*x*- axis, denoted as *x*-data on the figures, represents the number of nodes; *y*-axis, denoted as *y*-data on the figures, represents the number of tasks; whereas *z*-axis, denoted as *z*-data on the figures, represent time in ms). The presentation is done in 3D as the obtained time depends on two variables: the number of nodes and the number of tasks.

[28] Sigma Plot is used for obtaining figures.

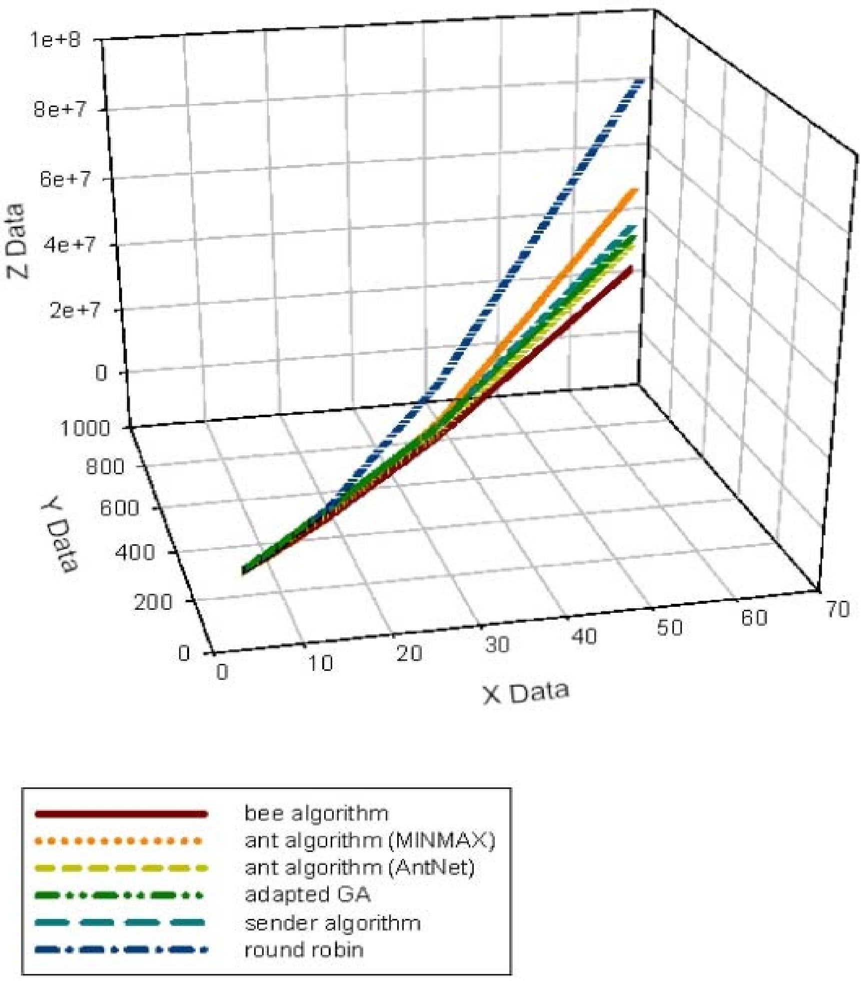


Figure 6.17. Algorithms Comparison (the Cluster environment)

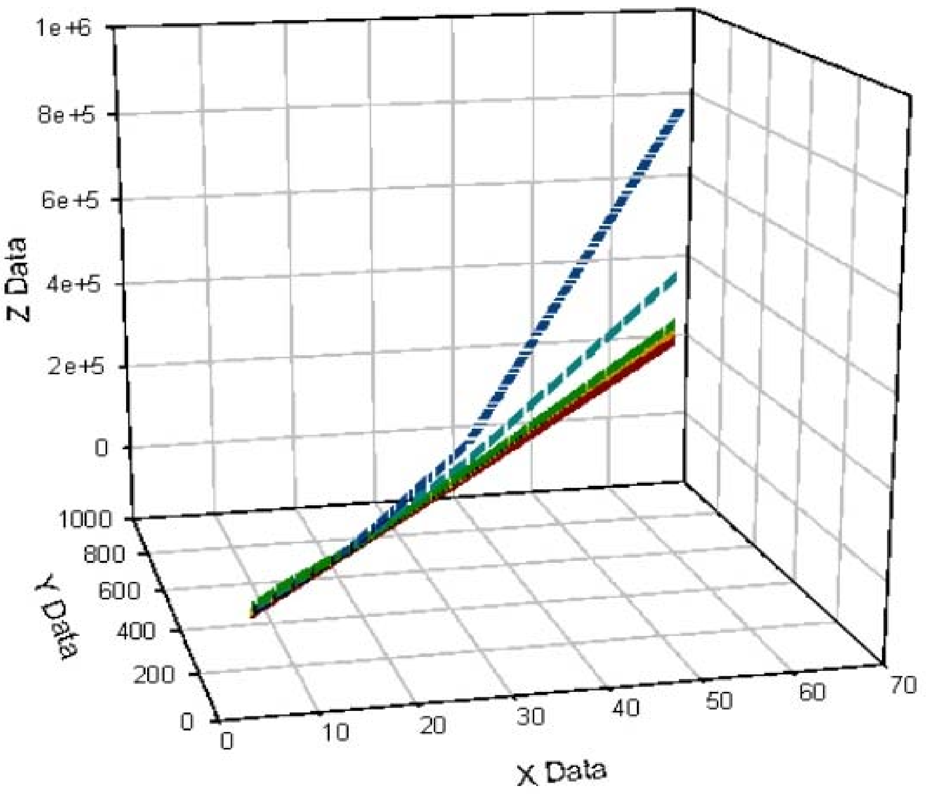


Figure 6.18. Algorithms Comparison (the Cloud environment)

The results that demonstrate the behaviour of the SILBA when the parameters, task size and the frequency of supplying tasks, change are shown in Table 6.8 and Figure 6.19, and in Table 6.9 and Figure 6.20, respectively. The used algorithm was MMAS, and the network of 16 nodes when supplying 200 tasks (in order to be in conformity with Table 6.5).

Table 6.8. Different task size.

task size (TS)	time (in ms)
TS1 (10kB)	50000
TS2 (20kB)	66000
TS3 (40kB)	69000

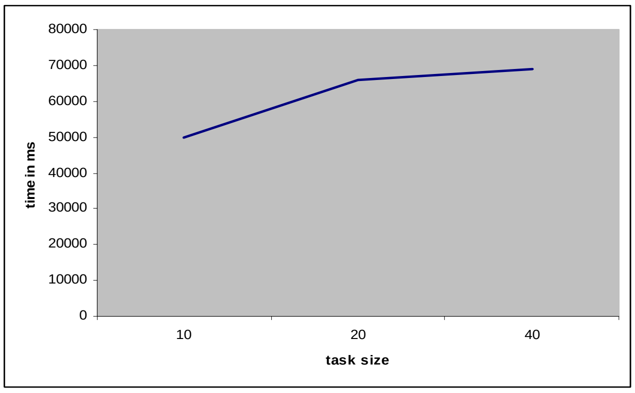


Figure 6.19. Different task size

Table 6.9. Different frequency of supplying tasks.

frequency	time (in ms)
All tasks are supplied at once	74000
Tasks are supplied in two waves: at the beginning and after 10 sec.	84000
Tasks are supplied in two waves: at the beginning and after 20 sec.	95000

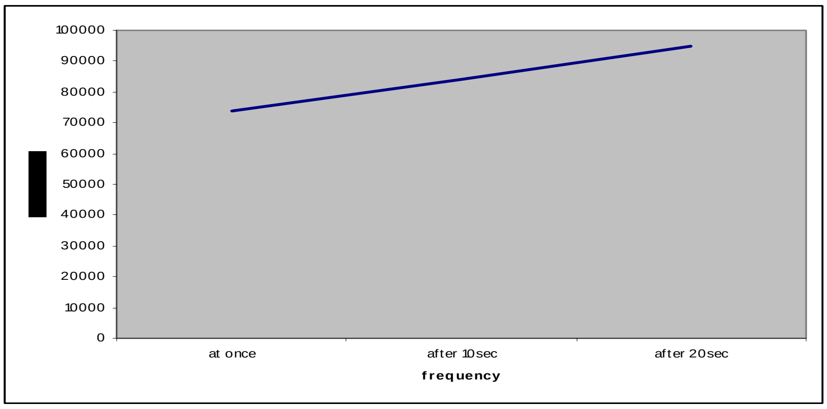


Figure 6.20. Different frequency of supplying tasks

Evaluation

Using the *absolute execution time* as metric for the benchmarks, bee algorithms showed the best results compared with the other algorithms ([ŠeKü10b], [ŠeKü10c]). On the cluster, the bee algorithm on 64 nodes performs 43% faster than MMAS, 19% faster than AntNet, 24% faster than Adapted GA, and 29% faster than Round Robin. In the more realistic environment of the Amazon EC2 Cloud, the bee algorithm on 64 nodes is 5% faster than MMAS 10% faster than AntNet, 13% faster than Adapted GA, 37% faster than Sender, and 68% faster than Round Robin. Also, MMAS ant algorithm performed well on the cluster. As stated in section 6.1., intelligent algorithms focus on important areas of the solutions space. Therefore, thir results are better compared with unintelligent ones. Especially, bee algorithm behaves well (better than others intelligent algorithms) – the algorithm simulates be behaviour in nature, i.e., bee software agent informs the "starting place" directly in a P2P way.

The values of the *suitability* function serve as a measure of self-organization and show how good the swarms are self-organized. The value of argument *x* (of the suitability function) serves to discern the usefulness of the intelligent algorithms and shows the correctness and adequacy of chosen partner nodes. It reflects how good the solution (i.e., the partner node) is chosen as well as the degree of self-organization of the used swarms. The average *x* value is 1, which means that the "best" node is always chosen.

The investigation about the task size shows that if the task size is double increased, the obtained time is slightly increased for smaller tasks (up to 20kb). This "slight" growth in time slows down for bigger tasks. Note that this investigation has nothing to do with the algorithm itself as the algorithm works in the same way when treating both big tasks and small tasks, and the number of tasks in this investigation is the same for each tasks size. This investigation shows the behaviour of the system in general. The absolute execution time *t* is defined as the makespan, but it comprises time needed for algorithm to be done. According to the obtained results, we can conclude that the bee algorithm behaves well and does not impose an additional complexity. Namely, the slight increased in time is the consequence of the increased complexity of the tasks needed to be executed.

The investigation about the frequency of supplying tasks shows the best results when tasks are supplied at once, but also acceptable time (slightly increased) when tasks are supplied in certain intervals of time.

6.2.2 Extended SILBA Benchmarks

As the extended SILBA supports the multi-level load balancing strategy, the goal of these benchmarks was to exchange concurrently the algorithms on each level and to investigate the solution of load balancing with this complex strategy. In considered case, there are 2 levels on which load balancing is realized concurrently: between several subnets and inside each subnet. Different network topologies are taken in consideration. The tests are created and performed on the basis of the following criterions:

- As the success of a particular combination depends on a network topology, the objective was to find the best combination of algorithms for each of the well-known topologies (chain, full, ring, star);
- For all located best combinations in particular topologies, compare and analyze them;
- After obtaining the best combinations, perform the benchmarks on different network (and subnets) dimensions and evaluate the scalability issue.

These benchmarks demonstrate two main messages: 1) the flexibility of the SILBA pattern based framework by showing that load balancing problem could be easily treated in a more complex network structures with several subnets, 2) a detection of those topologies which could profit mostly of swarm intelligent algorithms (particularly bee algorithms).

Test Examples and Test Environment

Test examples are constructed taking into account the following issues: the combination of algorithms, different number of subnets and number of nodes per subnets, increased number of clients per each subnet, different topologies ([ŠeKü11]).

The <u>*combinations*</u> (36) <u>*of*</u> all <u>*algorithms*</u> on two levels (6 algorithms on 2 levels): $level_1$ denotes the used algorithm inside a subnet, whereas $level_2$ denotes the used algorithm between subnets; the *values* of the respective parameters are described in Table 6.5 and reused from the basic SILBA.

Table 6.10. Combinations of algorithms

level1 / level2	Bee Alg.	MMAS	AntNet	adaptedGA	Sender	Round Robin
Bee Alg.	1	2	3	4	5	6
MMAS	7	8	9	10	11	12
AntNet	13	14	15	16	17	18
adaptedGA	19	20	21	22	23	24
Sender	25	26	27	28	29	30
Round Robin	31	32	33	34	35	36

Different *number of subnets* and *number of nodes per subnets*:

Table 6.11. Distribution of nodes in subnets

total number of nodes	number of subnets	number of nodes in each subnet
16	4	4
16	8	2
32	4	8
32	8	4

Increased *number of clients* per each subnet:

In the basic SILBA, only one client was responsible for putting the tasks into the network. This produced a light to moderate loaded network. In the extended SILBA, the number of clients per each subnet is increased until the subnet becomes fully loaded: for a subnet of *n* nodes, the assigned number of client is *n/2*. Each client supplied the same number of tasks. The clients are symmetrically positioned in order to have fairly loaded subnet. The same parameter is used for all test runs.

Different *topologies*:

The combinations of algorithms are tested on the well-known topologies: ring, star, full, chain. The objective was to define which combination of algorithms fits the best for a particular topology. Figure 6.21 depicts one example of each topology. The subnets could be both with intersections and without intersections, but in both cases at least one node from each subnet must possess two types of routing agents in order to allow for the realization of different types of load balancing algorithms (inside a subnet, between subnets).

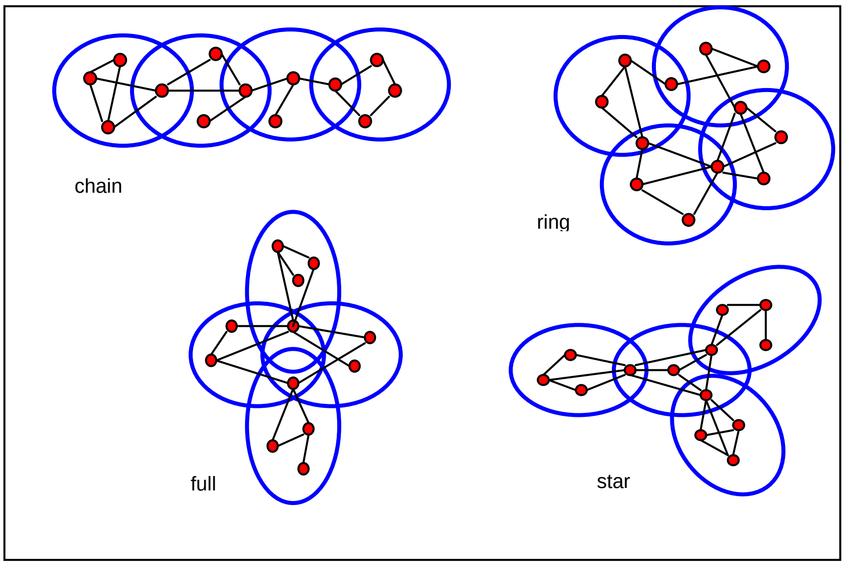


Figure 6.21. An example of different topologies: nodes are marked in red, subnets are marked in blue, possible connections are marked in black ([ŠeKü11]).

The same test environment, i.e., the Amazon Cloud, is used for performing these benchmarks. On the Amazon Cloud, we used the same standard instances as described in subsection 6.1.2.

Raw Result Data

The next figures (Fig.6.22 – Fig.6.25) show all combinations of algorithms on different topologies, searching for the best combination in each topology. The presented results demonstrate 4*4 structure, i.e., 4 subnets and 4 nodes in each subnet. In each subnet, each client supplied 200 tasks (i.e., the total of 1600 tasks), in order to be in conformity with the basic SILBA settings and results, and to be enough for achieving a heavy loaded network.

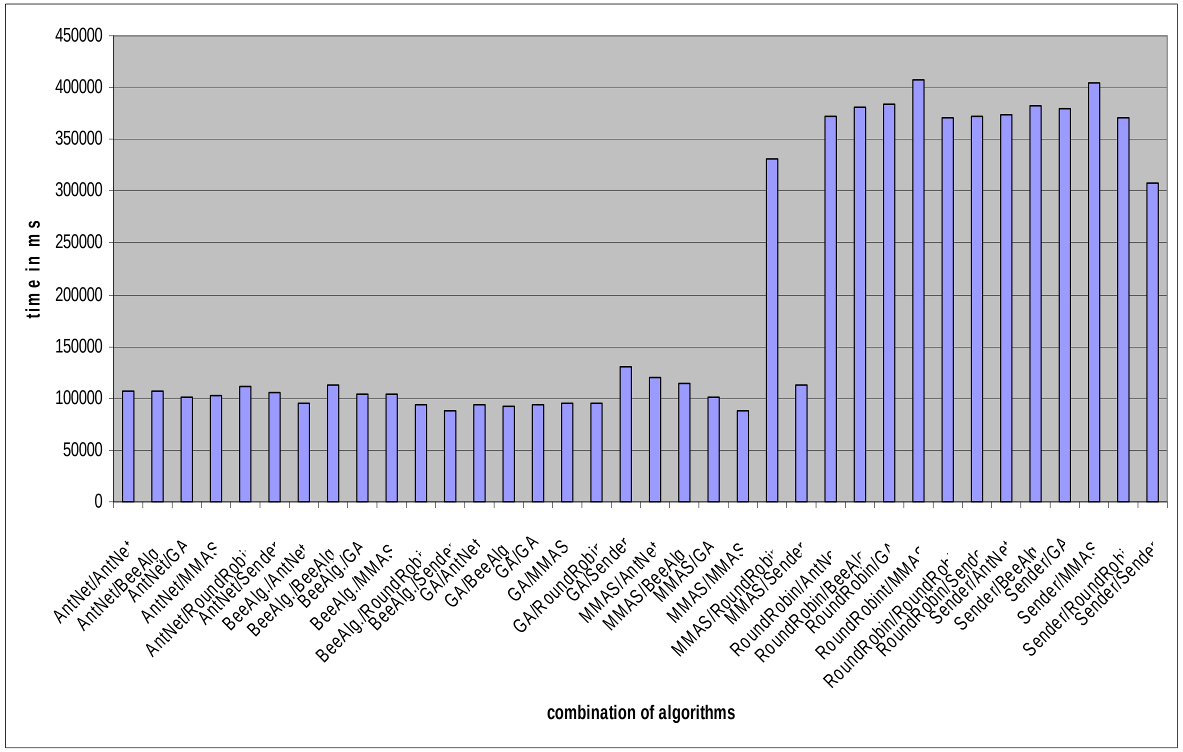


Figure 6.22. Combination of algorithms in the chain topology ([ŠeKü11])

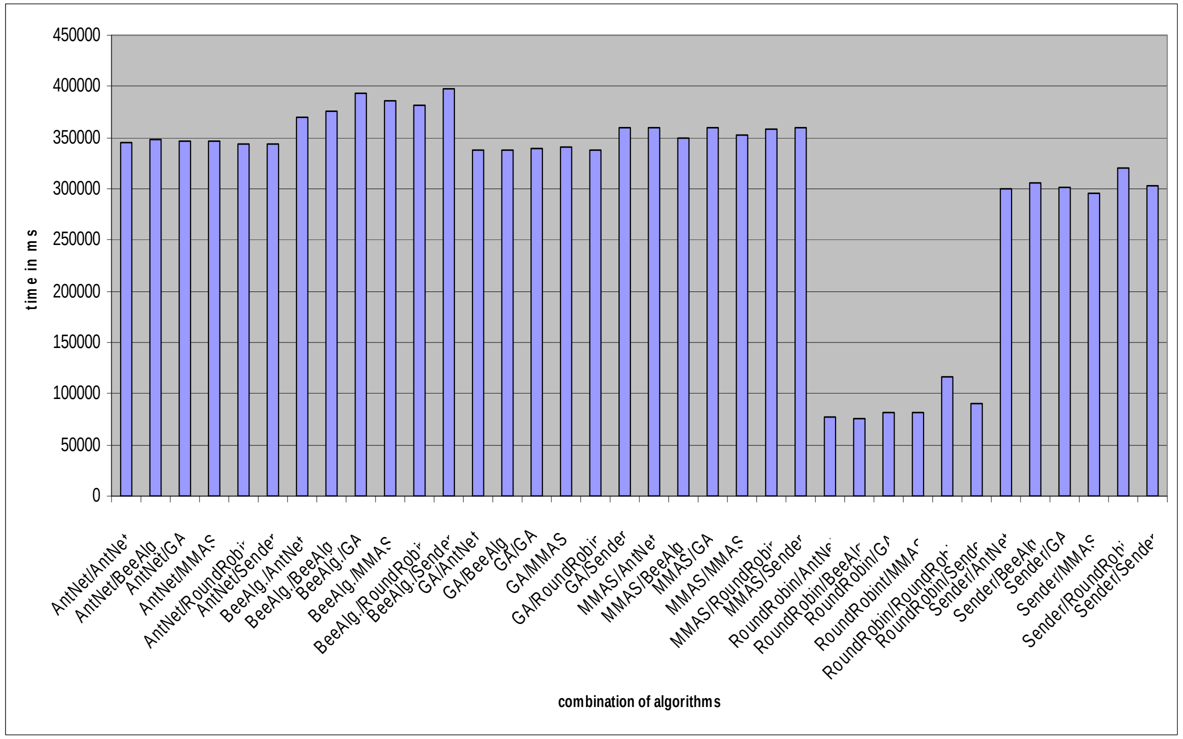


Figure 6.23 Combination of algorithms in the full topology ([ŠeKü11])

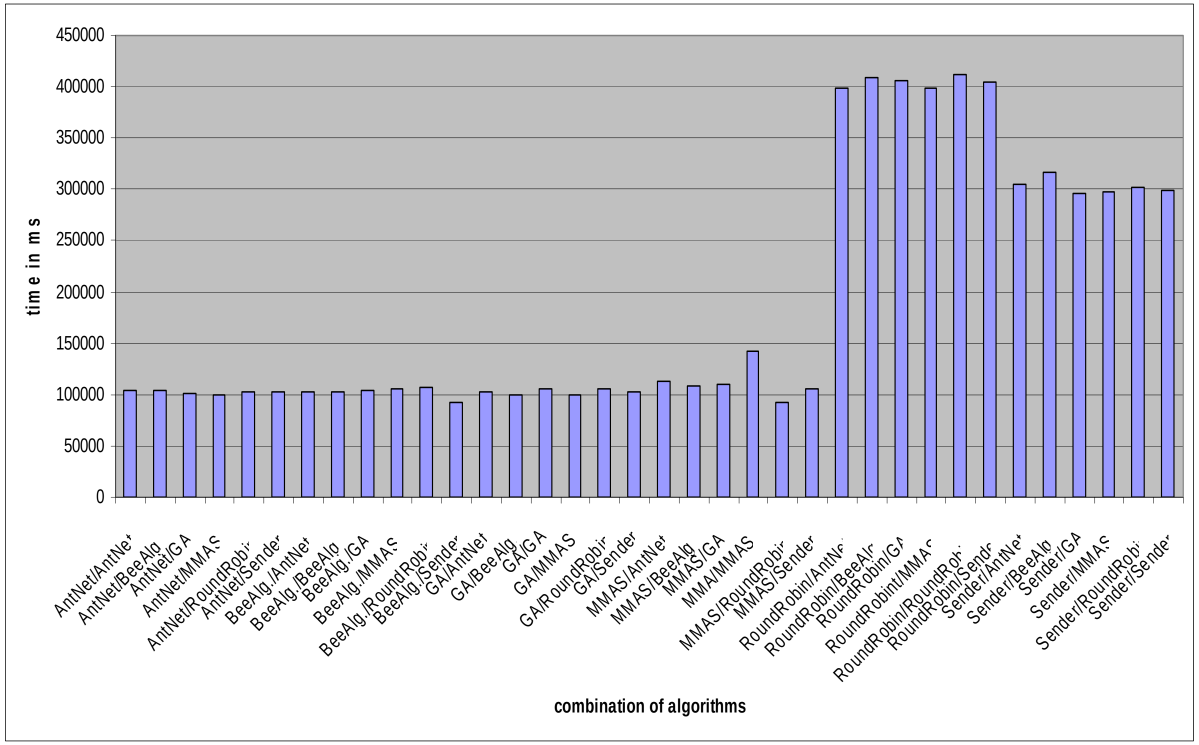


Figure 6.24. Combination of algorithms in the ring topology ([ŠeKü11])

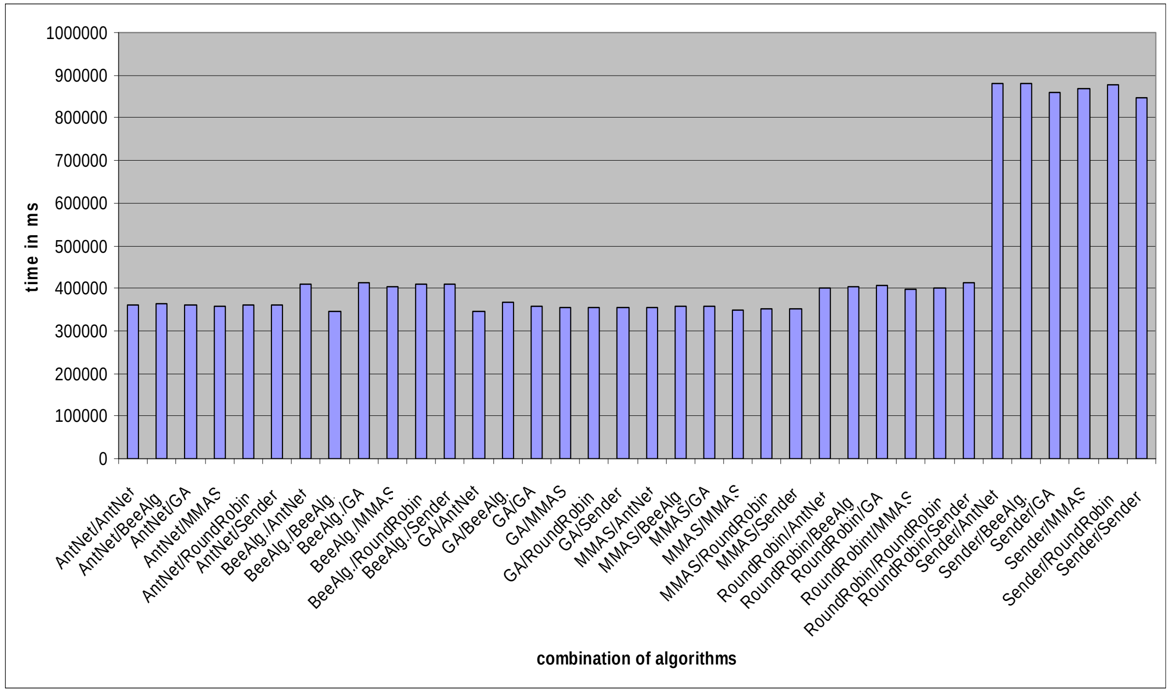


Figure 6.25. Combination of algorithms in the star topology ([ŠeKü11])

Table 6.12. shows the overall comparison made on the basis of the results obtained. Many appearances of the same topology in Table 6.12 denote that the respective combinations were equal good (e.g., both combinations BeeAlg./Sender and MMAS/MMAS were equal good in a chain topology). In almost each topology (except "star" topology), the best combination is made by one intelligent and one unintelligent algorithm. Although these combinations are not real hybrid algorithms (each pure algorithm works either inside subnet or between subnets), the overall load distribution in the whole network is realized through their synergy. Therefore, the intelligent algorithms find the good starting solutions (quality and fastness), while unintelligent algorithms could improve these solutions (fastness).

Table 6.12. Overall comparison of the best results in all topologies

topology	combination of algorithms	time (ms)
chain	BeeAlg./Sender	88000
chain	MMAS/MMAS	88000
full	RoundRobin/BeeAlg.	76000
ring	BeeAlg./Sender	93000
ring	MMAS/RoundRobin	93000
star	BeeAlg./BeeAlg.	346000
star	GA/AntNet	346000

The results from Table6.12 are presented in the next figure (Fig.6.26).

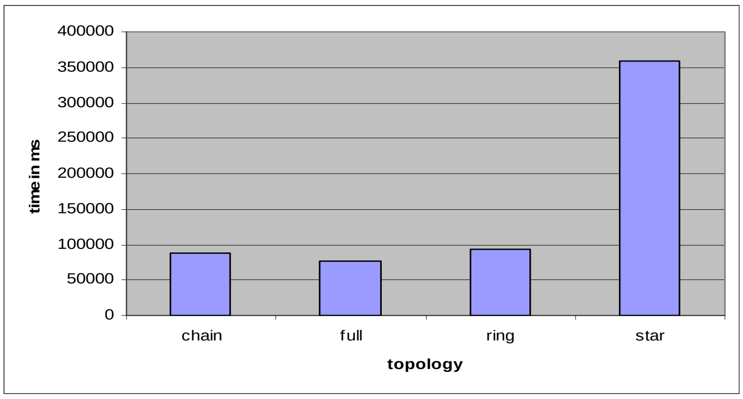


Figure 6.26. The results of the best combinations of each topology ([ŠeKü11])

After obtaining the best combination for each topology, the benchmarks with the best combinations are performed on larger network dimensions. Table 6.13 summarizes these results and shows that the results are stable as the same combination(s) of algorithms are obtained as the best ones for each of different dimensions (4*4, 8*2, 4*8, 8*4).

Table 6.13. Results of the best combinations in different network dimensions ([ŠeKü11])

total number of nodes	number of subnets	number of nodes in each subnet	Chain	full	ring	star
16	4	4	88000	76000	93000	346000
	8	2	374000	384000	359000	365000
32	4	8	420000	556000	582000	388000
	8	4	406000	455000	484000	356000

The extended SILBA offers better and more powerful solution than the basic SILBA ([ŠeKü11]). For example, in the network of 16 nodes: the best obtained results of the basic SILBA by processing 200 tasks is 60556ms, whereas the best obtained results of the extended SILBA by processing 1600 tasks (i.e., 8 times bigger load) is 76000ms.

The situations that can benefit from the extended SILBA are:

1. subnets are physically required, i.e., a given network is composed of a number of subnets
2. extremely large networks with highly increased number of nodes in which the building of subnets and applying the extended SILBA strategy could help in transferring load between very distant nodes; load needs not to be transferred via number of hops from one node to another one, so it could be transferred by using a shortcut, i.e., to "jump" from the subnet of its original node to the subnet of the distant destination node.

Overall Evaluation

The metric used in these benchmarks is the *absolute execution time*. According to the obtained results, it is obvious that the behaviour of a particular combination of algorithms depends on a topology. The analysis comprises the following issues ([ŠeKü11]):

1. How much the best combination (in each topology) is better than the "extreme" combinations: the worst one and the combination on the second place after the best one?

In order to answer to this question and perform the analysis, the numerical description expressed in percentages is used to distinguish the quality of combinations.

2. What is the "behaviour" of the other combinations, i.e., how much do they deviate from the best solution? What is the "collective behaviour" of algorithm combinations and the used SILBA framework in each topology?

The deeper analysis is done by using the additional measurements: the interval of variation and the root mean square deviation (RMSD). These measurements are introduced in order to examine the "behaviour" of the other combinations, i.e., how much they deviate from the best solution. The interval of variation is defined as the difference between maximum value of the used metric (time) and its minimum value: $t_{max} - t_{max}$. The used RMSD is a quantitative measure that tells how many good combinations in a particular topology exist, i.e., how far from the best solution the data points (the rest of the combinations) tend to be (smaller RMSD means more good combinations).

In the *chain* topology, the best result is obtained by both BeeAlgorithm/Sender and MMAS/MMAS. They were equal good, and better than the combination that "took the second place", GA/Bee Algorithm, for 5.4%, better than the worst combination for 78%, better that the average of all combinations for 56%. The interval of variation, $t_{max} - t_{max}$, is 320000ms. In the chain topology, the value of RMSD is 172121.

The combination RoundRobin/BeeAlgorithm showed the best results in the *full* topology. This combination was better than the combination that "took the second place", RoundRobin/AntNet, for 1.3%, better than the worst combination for 80.9%, better that the average of all combinations for 74.9%. The interval of variation, $t_{max} - t_{max}$, is 322000ms and the RMSD is 248227.7.

Both BeeAlgorithm/Sender and MMAS/RoundRobin were equal good in the *ring* topology. They were better than the combination that "took the second place", MMAS/RoundRobin, for 1.4%, better than the worst combination for 60.7%, better that the average of all combinations for 24.3%.

The interval of variation, $t_{max} - t_{max}$, is 535000ms and the RMSD is 216194.9.

In the *star* topology, the combinations BeeAlgorithm/BeeAlgorithm and GA/AntNet were the best with the same resulting value. They were better than the combination that "took the second place", AntNet/MMAS, for 6.1%, better than the worst combination for 77.4%, better that the average of all combinations for 50.1%. The interval of variation, $t_{max} - t_{max}$, is 319000ms and the RMSD is 153859.9.

Bee algorithms play a significant role in almost each topology, as the best obtained results in each topology are based on bee algorithms either used inside subnets or used between subnets or both. The rest of intelligent algorithms also gave good results in all topologies. The exception is the full topology where the best results are obtained when round robin algorithm is used inside subnets and combined with all others algorithms (except the combination Round Robin/Round Robin).

From numerical values of the RMSD, the greatest deviation is reached in the full topology. The majority of the other combinations differentiate a lot (they are worse in a significant extent) comparing to the best obtained combination. The smallest deviation is in the star topology, so the combinations behave evenly in this topology.

Another point of view is the analysis of how good response will be obtained by plugging any (random) combination of algorithms in the SILBA. The equally good results will be obtained in the star topology. So, the SILBA framework is very stable (without peaks in results) in the star topology. From the other side, Figure 6.26. shows that the results of the individual combinations of the SILBA pattern are successful for the chain, full and ring topologies, whereas the results obtained for the star topology are not so good.

In the next table, the behaviour of the swarm intelligent algorithms' combinations is extracted as these algorithms are promising ones and not so much exploited. Table 6.14 shows how much they deviate from the best solution in each of the used topologies. For example, the set of all combinations that use bee algorithms inside subnets is denoted in the table as "bee/others". According to these results, all combinations from this set deviate slightly from the best combination in the chain topology (that are BeeAlgorithm/Sender and MMAS/MMAS), whereas the combinations from this set deviate more from the best combination in the star topology, although the best combination is BeeAlgorithm/BeeAlgorithm.

Table 6.14 Deviation swarm based algorithms' combinations from the best solution ([ŠeKü11])

	chain	full	ring	star
RMSD (Bee/Others)	35171.0	755211.9	25337.7	141470.8
RMSD (Others/Bee)	417868.4	600503.1	387401.6	537938.7
RMSD(AntNet/Others)	44899.9	659335.3	25869.2	35787.1
RMSD(Others/AntNet)	404891.3	608559.9	372385.6	541636.4
RMSD(MMAS/Others)	249164.6	686738.7	58813.3	21725.6
RMSD(Others/MMAS)	450334.3	603189.0	371052.6	526899.4

Both hybrid algorithms and combinations of algorithm increase the probability of improving the solutions obtained by only one type of algorithm. For example, unintelligent algorithms suffer from the problem of finding good starting solution - these solutions are provided intelligent algorithms.

Besides the absolute execution time, the scalability is analyzed ([ŠeKü11]). The focus is on the issue of *load scalability*. A very general definition of scalability is taken into account ([JoWo00], [VaVS98]). Namely according to ([JoWo00]), a very general family of metrics can be based on the following definition:

$$\psi = \frac{F(\lambda_2, QoS_2, C_2)}{F(\lambda_1, QoS_1, C_1)} \tag{6.2}$$

where *F* evaluates the performance, λ evaluates the rate of providing services to users, *QoS* is a set of parameters which evaluate the quality of the service seen by users and *C* reflects the cost. Further, (Jogalekar and Woodside 2000) establishes the scaling strategy by means of a scaling factor *k* and the set of scaling variables which are functions of *k*. They express the strategy as a scaling path in a space in which they are the coordinates. In ([JoWo00]), it is possible to see how $\psi(k)$ might behave in different situations (Fig.6.27). It is already introduced a simplified version of interest to this problem (Eq. 6.1) in terms of load, resources and performance measure.

In the presented benchmarks, the increasing of load concurrently with the increasing of the resources is applied and analyzed. By comparing results (Tables 6.6. and 6.7 for the basic SILBA, and Table6.13 for the extended SILBA), it is easy to see that the best chosen combinations based on bee algorithm scale well ([JoWo00]). Load and resources are increased *twice* for consecutive test runs.

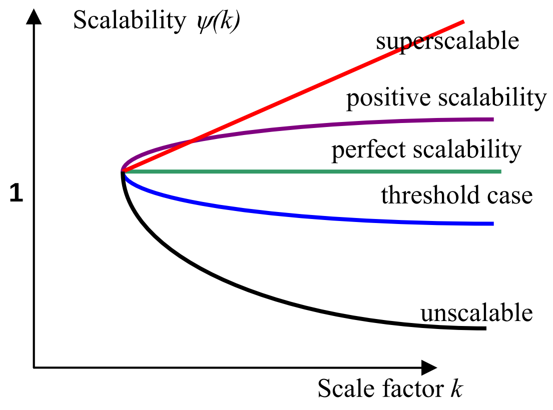


Figure 6.27. Scaling behavior ([JoWo00])

Scalability in the basic SILBA

For example in the cluster environment, load and resources are increased twice for consecutive test runs, i.e., they are increased by 2^n compared with the starting test run (4 nodes, 50 tasks). The values of r_{scal} are 2.9, 3.0, 3.4, 3.2 (rounded to one decimal) for consecutive bee test runs, i.e., 2.9, 9.1, 31.0, 100.8 compared with the starting test run (4 nodes, 50 tasks). These values converge to positive scalability. Such behaviour is even better in a more real environment, i.e., on the Cloud. Almost the similar situation occurs with AntNet algorithm.

Scalability in the extended SILBA

In the *chain* topology:

a) if the number of subnets are increased and the number of nodes inside a subnet is the same, i.e., **4***4, **8***4, r_{scal} is 4.6 (rounded to one decimal), that leads to positive scalability;

b) if the number of nodes in a subnet is increased and the number of subnets is the same, i.e., 4***4**, 4***8**, r_{scal} is 4.8; 8***2**, 8***4**, r_{scal} is 1.1, that converges to perfect scalability.

In the *full* topology:

a) if the number of subnets are increased and the number of nodes inside a subnet is the same, i.e., **4***4, **8***4, r_{scal} is 5.98; that leads to positive scalability;

b) if the number of nodes in a subnet is increased and the number of subnets is the same, i.e., 4***4**, 4***8**, r_{scal} is 7.3; 8***2**, 8***4**, r_{scal} is 1.8; that leads to positive scalability.

In the *ring* topology:

a) if the number of subnets are increased and the number of nodes inside a subnet is the same, i.e., **4***4, **8***4, r_{scal} is 5.2; that leads to positive scalability;

b) if the number of nodes in a subnet is increased and the number of subnets is the same, i.e., 4***4**, 4***8**, r_{scal} is 6.2; 8***2**, 8***4**, r_{scal} is 1.3; that converges to perfect scalability.

In the *star* topology:

a) if the number of subnets are increased and the number of nodes inside a subnet is the same, i.e., **4***4, **8***4, r_{scal} is approximately 1; that leads to perfect scalability;

b) if the number of nodes in a subnet is increased and the number of subnets is the same, i.e., 4***4**, 4***8**, r_{scal} is approximately 1; 8***2**, 8***4**, r_{scal} is approximately 1; that that leads to perfect scalability.

6.3 Summary

In this chapter, three different sets of results are presented:

1) The advantages of using intelligent lookup mechanisms in searching, locating and retrieving information. In this problem, the combination of one unintelligent approach (random positioning of data) and one intelligent approach (retrieving of data) obtained the best results. Antnet algorithm showed uniformly good results in both environments. Bee algorithm showed the best results on big instances.

2) The advantages of using bee swarm intelligence in the context of load balancing are presented. Additionally, two further intelligent algorithms are adapted based on MMAS and AntNet ant algorithms. For these three algorithms, the best combination of feasible parameters was identified. For the comparison and evaluation a generic load balancing framework that allows the plugging and thus easy exchange of algorithms was used. The three intelligent algorithms were compared with three well-known unintelligent algorithms, Round Robin, Sender, and Adapted Genetic Algorithm. The load was generated by one single client, and as performance parameter the absolute execution time was used. Under these conditions, the obtained results show that the bee algorithm outperforms all other test candidates.

3) The advantages of using bee swarm intelligence in the combination with the other algorithms (both intelligent and unintelligent) in solving load balancing problem in more complex network structures that consist of different subnets which might overlap each other and have nested structure. After investigation of different network topologies, the combinations that are based on swarm algorithms showed the best results in the chain, ring and full topologies. The best combinations in all topologies are based in bee algorithms. The load is generated by many clients, positioned symmetrically in subnets. The performance measure was the absolute execution time, expressed in milliseconds. The best obtained combinations scale good in all investigated topologies.

CHAPTER 7

7 CONCLUSION

Complex adaptive systems from our environment are intended "succeed" to overcome all turbulences and adapt to new circumstances. One of the most powerful mechanisms is the mechanism of self-organization. It triggered a significant attention in scientific research of different disciplines in the recent years. Self-organization needs to be understood in order to apply it. Self-* mechanisms are not yet completely understood and explained. The applications of self-* mechanisms offer the possibilities to cope with complexity in different scientific areas, including information technology.

An increased complexity in IT industry needs to be urgently treated in different, advanced ways. The application of self-* approaches to IT problems is a promising way to solve the problem of increased complexity.

This dissertation investigated the application of self-* on two well-known IT scenarios: dynamic load balancing in heterogeneous distributed systems, and location and retrieval of data in the Internet.

First, it discussed which IT problems are suitable for the application of self-* mechanisms and what kind of complexity can be found in these IT scenarios. A short classification is given and concluded that different types of complexity could be treated by self-* principles on a different ways. For example, if programming complexity is a problem, then some kind of methodology based on high decoupling and autonomy is needed to be applied. In case that computational complexity is a problem, then an algorithmic-heuristic support can help.

Also, the ways of measuring complexity is analyzed. As it is closely connected to the measurement of self-organization, different proposed methods were presented. However, it is firmly correlated to the problem of interest and therefore, some general measure of self-organization does not exist.

Further, appropriate software architectures were constructed and nature based self-* mechanisms were applied to the two above numbered IT problems. It could be concluded that for each self-* mechanism that should be mapped to the IT problem, some kind of modeling (usually mathematical) is needed as a prerequisite for the future designing of algorithms.

The problem in the first scenario and the achievements and contribution can be shortly described as follows. Nowadays global networks are overloaded with huge amounts of different information. Searching for data can be a time consuming and exhaustive process. This implies a necessity for the existence of an effective lookup mechanism. Therefore, the presented self-organizing approach that combines decentralized unstructured P2P with space based computing is constructed in order to effectively locate and retrieve information from a network. This approach differs from others according to several issues: it focuses on the quality of the solutions, including the time needed to obtain that solution, it is based on a learning principle from the nature, which helps for future searching trials, and the common disadvantage in the used basis-combined approaches – not so good scalability – can be diminished. E.g., this approach outperforms Gnutella lookup. It is demonstrated this by means of different benchmarks. For example, on the Amazon EC2 Cloud, the random/bee combination on 80 nodes with 50 swarms and by treating 5 queries was 0.5% better than the random/AntNet combination, 7.8% better than the random/MMAS combination and 61.3% better than Gnutella. This first presented approach is:

- self-learning, because of a learning capabilities of swarm collective intelligence,
- self-repairing and self-configuring, because of the used algorithms (AntNet algorithm is a typical representative) allow for adding/removing nodes (i.e., connections).

The problem in the second scenario considered dynamic load balancing in nowadays heterogeneous distributed networks. A workload should be evenly distributed across two or more computers, network links, CPUs, hard drives, or other resources, in order to get optimal resource utilization, maximize throughput, minimize response time, and avoid overload. The presented approach comprises self-organizing, adaptable, generic framework based on autonomous agents' collaboration by using space based computing technology that additionally emphasize autonomy and high decoupling of processes. Above all, different algorithms were plugged in (both intelligent and unintelligent ones). Two different set of results are presented through benchmarks.

The first set of results is obtained on the basic SILBA and presents the benefits of using bee swarm intelligence in the context of load balancing. Further, two intelligent algorithms based on ants behaviour, MMAS and AntNet ant algorithms, are adapted for load balancing. For these three algorithms, the best combination of feasible parameters was identified, and afterwards, these three intelligent algorithms were compared with three well-known algorithms, Round Robin, Sender, and Adapted Genetic Algorithm. The load was generated by one single client, and shows the situation in a system that is lightly to moderately loaded. The absolute execution time was used as the performance parameter. Under these conditions, the obtained results show that the bee algorithm outperforms all other test candidates. On the Amazon EC2 Cloud, the bee algorithm on 64 nodes was 5% faster than MMAS 10% faster than AntNet, 13% faster than Adapted GA, 37% faster than Sender, and 68% faster than Round Robin.

The second set of results is obtained on the extended SILBA and proves the advantages of using bee swarm intelligence in the combination with the other algorithms (both intelligent and unintelligent) in solving load balancing problem in more complex network structures that consist of different subnets possibly overlapped with each other and have nested structure. The combinations of algorithms are tested on different network topologies: chain, ring, full and star. The combinations that are based on swarm algorithms showed the best results in the chain, ring and full topologies. The best combinations in all topologies are based in bee algorithms. The load is generated by many clients, positioned symmetrically in subnets. To sum up, in the *chain* topology, the best result is obtained by both BeeAlgorithm/Sender and MMAS/MMAS. They were equal good, and better than the combination that "took the second place", GA/Bee Algorithm, for 5.4%. The combination RoundRobin/BeeAlgorithm showed the best results in the *full* topology. This combination was better than the combination that "took the second place", RoundRobin/AntNet, for 1.3%. Both BeeAlgorithm/Sender and MMAS/RoundRobin were equal good in the *ring* topology. They were better than the combination that "took the second place", MMAS/RoundRobin, for 1.4%. In the *star* topology, the combinations BeeAlgorithm/BeeAlgorithm and GA/AntNet were the best with the same resulting value. They were better than the combination that "took the second place", AntNet/MMAS, for 6.1%. The presented situation refers to a fully (highly) loaded system. The performance measure was the absolute execution time, expressed in milliseconds. The load scalability is additionally investigated. The best obtained combinations scale well in all investigated topologies. This second presented approach is:

- self-learning, because of a learning capabilities of swarm collective intelligence,
- self-repairing and self-configuring, because of the used algorithms (Antnet algorithm is a typical representative) allow for adding/removing nodes (i.e., connections),

- self-tuning, because it provides a support for coping with high and dynamically changing loads through load balancing.

Both architectures constructed for the numbered scenarios are modelled and specified by using the PlusCal algorithm language. Their correctness is justified, i.e., it is proven that they are correct for each combination of algorithms, all policies and all topologies.

This investigation can be extended by taking in consideration different metrics used for the evaluation of results (e.g., communication delay, utilization, stability, fairness across multi-user workloads, robustness in the face of node failure, adaptability in the face of different workloads, etc.), a benchmarking of large instances, an investigation of the impact of load injection in different places in the network and a building of the recommendation system for a given problem (e.g., the determination of the best topology, algorithm combinations, parameters tuning, etc. for a particular problem).

The starting research questions have therefore been answered as follows in the dissertation:

1. **Research Question 1:** *Can the two important IT use cases: 1) load balancing in heterogeneous distributed systems, and 2) information retrieval in the Internet, profit from the usage of self-organization?*

The presented scenarios had profit of the usage of self-organization. In chapter 6, the used metrics (absolute execution time, scalability) are improved, present the benefits of using self-organization in these scenarios and show the applicability of self-organization to these types of problems. Both scenarios - load-balancing in heterogeneous distributed systems, and searching, retrieving as well as placing information in the Internet - are suitable for the appliance of self-mechanisms as they contain NP hard problems that searching and optimization. In chapter 6, it is described how and in which extent the employed principles of self-organization improve performance and scalability, that bee intelligence outperformed other (un)intelligent approaches taking in consideration the quality of a solution, the metric used and a scalability issue.

2. **Research Question 2:** *Can the principles of self-* help to cope with complexity in heterogeneous systems? What can be improved by employing self-* mechanism? What kind of complexity exists and how can complexity be measured with the focus on the above mentioned problems?*

The principles of self-* helped to cope with complexity (programming and computational) in heterogeneous systems due to a treatment of the presented IT scenarios. The performance and scalability are improved. In these specific cases, the degree of self-organization in the systems is measured by means of the similarity, i.e., suitability function that showed how much software agents (swarms) are organized. In both cases, this function took the highest value that means the highest quality of data was obtain, i.e., the self-organization in the system is measured as very high.

3. **Research Question 3:** *How can swarm intelligence be mapped/adapted to the load balancing problem and to the problem of locating and retrieving information in the Internet? Can bee intelligence be mapped to these two use cases and how? Can ant intelligence be adapted to these two use cases and how?*

As it is already stated, the mapping and adapting of swarm intelligence is done by using mathematical models, and consequently by the appropriate algorithms. Chapter 5 addresses research question 3 by describing how swarm intelligent can be mapped or adapted to the located application cases. Ant intelligence required remodeling due to the specific problems, whereas bee intelligence required inventing new parts in the recruitment phase and modeling bee algorithm for a specific problematic. In this case, the novelty is the implementation of bee intelligence for the load balancing problem for the first time in order to improve the quality of the solution and scalability.

4. **Research Question 4:** *What is the best parameters tuning in each of the considered scenarios?*

The fine-tuning of parameters and the best parameters' sets are discovered experimentally (through benchmarks) as the considered algorithms are non-deterministic, and presented in chapter 6. Due to these parameters' settings, swarm intelligence outperformed other (un)intelligent approaches taking in consideration the quality of a solution, the metric used and a scalability issue.

5. **Research Question 5:** *Is it better to have an intelligent approach or an unintelligent approach or a certain combination (which one)?*

The combination of intelligent and unintelligent algorithm showed the best results. Unintelligent algorithms suffer from the problem of finding good starting solution that can be provided by the swarms. In chapter 6, it is detected which combination of algorithms fits the best to a particular network topology; also, detection which topologies profit the most from the application of swarm intelligence (by means of the used metric and scalability).

7.1 Future Work

Future work will be concentrated on further developing self-* oriented solutions in overcoming complexity in IT. Some of the points that will be researched are:

- Investigation of new areas of natural inspired self-* that are not exploited enough or not exploited at all (e.g., like bark beetles, slime mold, fireflies, etc.) and mapping them to real-world problems.
- Implementing self-* features in a software system in order to additionally self-improve itself after a certain period of time. A further challenging objective is to "upgrade" the notion of self-* as follows: a self-* system is periodically subjected to a process of evolution that should be triggered automatically. The goal will be to make this process applicable to different scenarios and independent of a specific problem, and to development instruments for "system evolution", i.e., system designs shall incorporate "evolution tools" in their origin, so that a complete system can be upgraded and changed with a low negative impact on the environment as a whole. The intention is to specify necessary theoretical and algorithmic frameworks for a self-evolving infrastructure that is able to maintain itself using its own mechanisms. In this scope, self-evolution is not one further self-* property. Rather this is a complex process which a self-* system is subjected to through its life time. No doubt that systems change over time. The notion of "evolution" will denote creative changes that lead to an improved system. The vision is to apply this idea to already self-* systems as well as to any other kinds of systems, and to be completely general in order to be applicable to a wide set of application problems. Over time, a successful self-evolving system will be able to evolve more effectively. The future work will emphasize that self-evolving systems and self-* systems are not the same. When a system selects for structures that enable it to evolve more effectively, a meta-selective process emerges within that system to direct the system's own development. Self-evolution could be realized on a meta level. The intention is to develop a general framework and to incorporate it in our already created self-* systems (like a proof-of-concept) in order to obtain a complex composition of self-(self-)* systems.

APPENDIX

8 APPENDIX

Appendix A

The Appendix A contains some characteristic implementation details of both scenarios (information retrieval and dynamic load balancing) carried out in Java.

The implementation details of Information retrieval scenario

The following part gives more details about the implementation of the information retrieval scenario by putting emphasize to the ant algorithms. The plugging of bee algorithm is similar from the architectural point of view.

The main class is **AntNode**. The methods for processing the content of ASNode and ACNode containers are defined in AntNode class. For every container, there is a listener – so, every time when some content in the container is created, the appropriate method is activated: p*rocessMessagesEntries(), processRouteEntries(), processAntEntries().* After a content restoring, they start methods for processing of a restored object: *processmessage(), processRoute(), processAnt()*. Except these methods, the methods for writing a content in the containers of the other nodes are necessary: *putMessageEntry(), putRouteEntry(), putAntEntry().* The methods for messages processing and paths processing are specialized in the inherited classes ASNode and ACNode. **ASNode** includes some special containers: *Message Container, Route Container, Ant Container* and *Content Container*. *Message Container* and *Route Container* are used in the initialization of the system. *Content Container* is lookup container, which content is the subject of search. **ACNode** serves for initialization of the system by sending the message REGISTER to the Message Container of every particular ASNode, and afterwards, by sending the Routing Table into the Routing Container of every particular ASNode. This is implemented in *run* method of this class. Some steps of run method are defined in the inherited classes **MMASAntColony,** i.e., **AntNetAntColony** (Fig.8.1**)**. Procedure *registerNodes()* initializes every ASNode in the network by adding to it an unique *id*. Then REGISTER message is written in Message Container of the appropriate node; afterwards ASNode processes that message by *processMessage().* The next step is to send *i*th row of “distant” matrix to *i*th node in the network. ASNode processes REGISTER message that carries its *id* in the network and the address (URI) of ACNode. After this message, then accepts the routing table and creates NNList (nearest neighbours). Similarly, the methods for messages sending and paths sending are specialized in the inherited classes ASNode and ACNode.

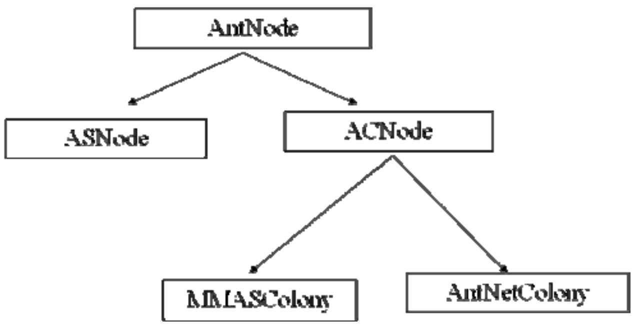


Figure 8.1. The organization of the main classes in Lookup scenario.

The ant is implemented by **Ant** class that possesses the attributes (among the usual ones):

- node – the current node where the ant is (it takes the routing table from this node);
- content – the content that the ant leaves on the suitable node (putAnt) or the ant searches for on the network (getAnt);
- status – an indicator: Prepare, ForwardAnt, BackwardAnt;
- actionStatus – an indicator of how the ant action was successful: no_data, acceptable_data, exact_data;

Ant class defines the ant behaviour on the node. When the ant is on a node, then a node initiates its *enterNode* method that starts (among others actions) its run method and run method implements the ant behaviour. The ant on ASNode performs *runForward* procedure or *runBackward* procedure, depending on its status. In *runForward*, the ant calculates the used time (current time – starting time from the previous node), sets the values of the arrays *tour* (the array that describes the travelled path) and *visited* (the array of visited nodes), checks local lookup container by *checkLookupSpace()* that is defined in inherited classes MMASPutAnt, MMASGetAnt i.e. AntNetPutAnt, AntNetGetAnt, depending on the types of ants (putAnt leaves the content on the suitable node, while getAnt searches for acceptable content on the network). After this check, it changes the status (becomes backwardAnt and starts with *runBackward* procedure) or continues further - chooses the next node by using *decisionRule(*) method and leaves the current node by using *leaveNode()* method. In *leaveNode()*, it notes the current time and calls *putAntEntry()* methods of the current node that enables the ant to go through the network on the chosen node where it repeats the procedure. In *runBackward*, it returns through the same way and updates (optionally) the routing table of the current node. The routing table contains data about the length of the path, the amount of pheromone, …). At the end, it comes back on the ACNode (it started from this one) and runs one of these methods (depending on the performed action status): *runNoData(), runExactData(), runAcceptableData()*. In every of these method, the path is optimized by using *localsearch()* method.

ACNode performs the creation of ants (*createAnts()*) and sends them on the network (*startIteration*). Upon finishing the iteration, ACNode informs the other nodes in the network about that by sending the message FINISH. Then all nodes return possible updated routing tables (it is implemented in *getChanges()*). After that, ACNode starts method *updateStatistics().* The classes **ASNodeStarter** and **ACNodeStarter** start the system. ASNode-Starter starts ASNode (it assigns the port on the local address), while AC-NodeStarter starts ACNode for the given number of ASNodes, assigns to it the local port, connects the nodes in the network.

More information about the some of the main classes can be found in the following part:

Ant

It implements an ant. The ant on the node in the network is doing its *run* method. This is the base class that is specialized by classes MMASAnt, AntNetAnt.

```
private int id;              //id of ant
private Object node;         //current node
private int step;            //current step on the travel
private int finalStep;       //end step
private  long  tourLength;    //total  length  of  the  travel
                               passed
private int[] tour;          //current travel path
private boolean[] visited;   //visited nodes
private AntStatusType status; //status of the ant
private Content content;     //the content that is to be put
                              i.e. search
private AntActionStatusType actionStatus; //action
private long timeOut;       //time-to-live
private long startTime;    //time of starting of the ant to
                              the node
```

The ants are going in parallel and there is a thread for every ant. The method runAtNode implements cases: runForward and runBackward in a sense of getStatus. The method runAtACNode implements NoData, AcceptableData, ExactData, Error in a sense of getActionStatus.

AntActionStatusType

It gives the current status of the action that an ant is doing (NoData, AcceptableData, ExactData, Error).

AntStatusType

The current status of the ant: PREPARE status – with this status the ant is created; when the ant goes to the network, status FORWARD is taken.

AntNode

The node where an ant may be present -the main class of the nodes in the network.

ASNode

The node in the network which content is the subject of ant searching.

ACNode

It initializes the system and starts a series of iterations. During the iteration, a certain number of ants is created and sent to the network, and then some global changes are done (*run*-method). This *run*-method comprises: startIteration, finishIteration, getChanges, UpdateStatistics, GlobalPheromoneUpdate.

CNNAnt

The ant that is going through the partial path on the network by using NearestNeighbor algorithm. They are used for determining the maximal partial path in order to determine the initial value for tau0**.**

Content

The content the lookup table and the same time the content that the ant is carrying on and uses for searching. Also, the similarity function is implemented here.

Instance

The structure of the network for Ant Algorithms. NNList, choice_info and heuristics are calculated here.

```
public class Instance {
    public static double ALPHA=1.0;
    public static double BETA=2.0;
    public static double XI=0.1;
    public static double RO=0.5;
    private int n;           //the number of nodes
    private int[][] dist;    //distance matrix
    private int nn;          //length of NNList
    private int[][] nnList;        //array of the nearest
    neighbours
    private double[][] pheromone;   //pheromone matrix
    private double[][] choiceInfo; //combination of phero-
    mones and  heuristic
```

RouteTo
The edge of the graph with the information about the address of the ending node (URI uri).
Route
The edge of the graph with the information about the ends.
AntNetAnt
The specialized ANT class that implements AntNet algorithms.
AntNetAntColony
The specialized ANT class that implements AntNet specific algorithms.
AntNetGetAnt
Get Ant in AntNet algorithm.
AntNetPutAnt
Put Ant in AntNet algorithm.
AntNetInstance
The structure of the network for Antnet algorithms.
KOpt
Local Search algorithms
MMASAnt
The specialized Ant class that implements MMAS algorithms.
MMASAntColony
The specialized class of ACNode that implements MMAS specific algorithms.
MMASInstance
The structure of the network for MMAS algorithm.
MMASGetAnt
Get Ant in MMAS algorithm.
MMASPutAnt
Put Ant in MMAS algorithm.

APPENDIX

The implementation details of dynamic load balancing scenario

Some main parts of the SILBA implementation are shortly described here. The abstract SilbaBase class is created with the following inheritance:

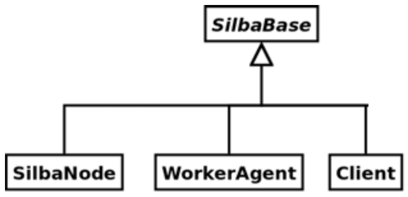
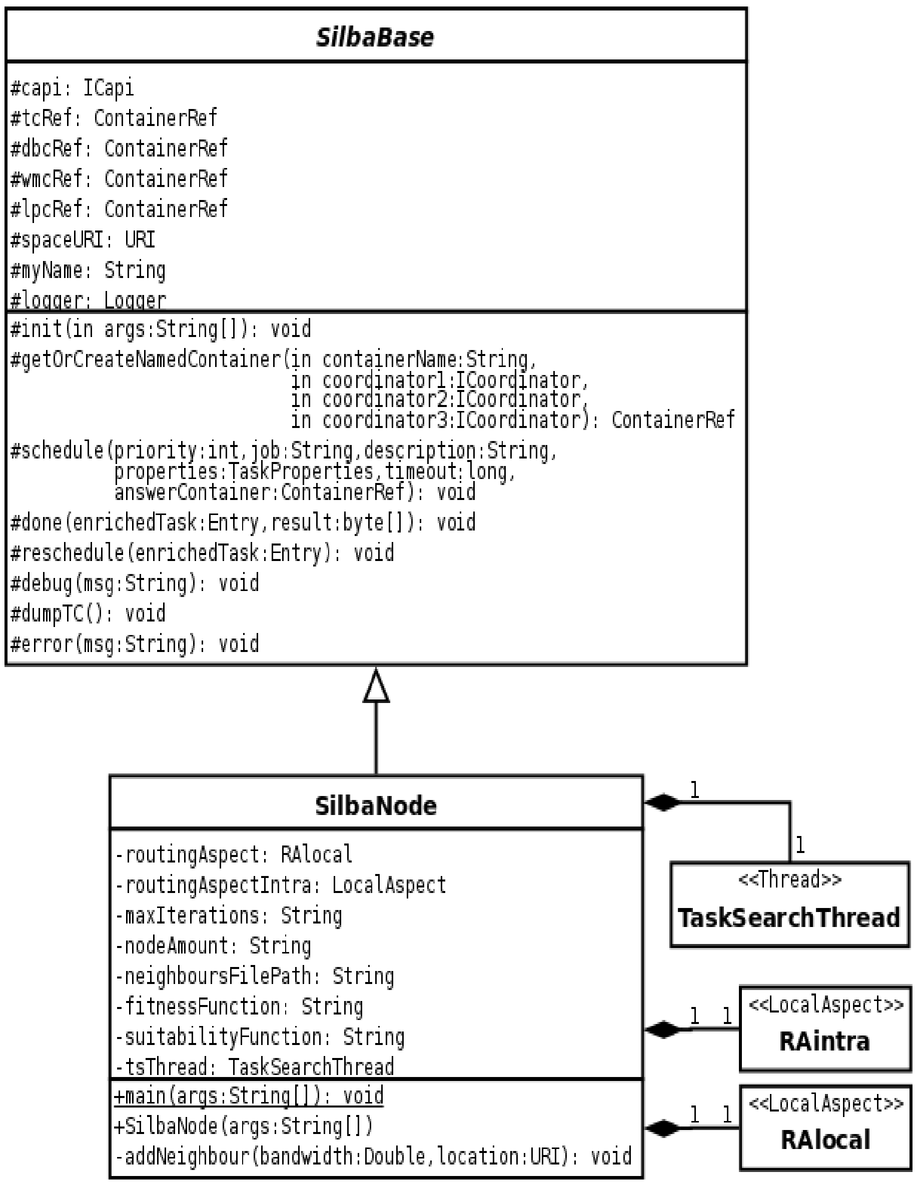


Figure 8.2. The organization of the main classes in SILBA.

SilbaNode

SilbaNode itself is a simple class. It sets up the containers, takes arguments from the command line which are then passed to the routing agent. It also adds the routing agents as aspects of the task, reads the file containing routing information and writes that information into the routing space. For example, if a search from an underloaded node should be conducted, the SilbaNode also starts a thread that regularly checks this node's load status and, if the node is underloaded, writes a request for a search for suitable tasks. SilbaBase requires the following arguments on start up:

- the path to the MozartSpaces properties file
- the TCP/IP address of the node itself (this is the address that other agents will also use to connect to the containers)
- the name of the node
- the path to the neighbors file
- the number of iterations the algorithm should perform at the most
- and the number of nodes in the setup, thus determining the number of swarms that should be used in the algorithm
- a value between 0 and 3 determining which of four suitability functions is being used and
- a value between 0 and 3 determining which of four fitness functions is being used.



SilbaBase
#capi: ICapi
#tcRef: ContainerRef
#dbcRef: ContainerRef
#wmcRef: ContainerRef
#lpcRef: ContainerRef
#spaceURI: URI
#myName: String
#logger: Logger
#init(in args:String[]): void
#getOrCreateNamedContainer(in containerName:String, in coordinator1:ICoordinator, in coordinator2:ICoordinator, in coordinator3:ICoordinator): ContainerRef
#schedule(priority:int,job:String,description:String, properties:TaskProperties,timeout:long, answerContainer:ContainerRef): void
#done(enrichedTask:Entry,result:byte[]): void
#reschedule(enrichedTask:Entry): void
#debug(msg:String): void
#dumpTC(): void
#error(msg:String): void
SilbaNode
-routingAspect: RAlocal
-routingAspectIntra: LocalAspect
-maxIterations: String
-nodeAmount: String
-neighboursFilePath: String
-fitnessFunction: String
-suitabilityFunction: String
-tsThread: TaskSearchThread
+main(args:String[]): void
+SilbaNode(args:String[])
-addNeighbour(bandwidth:Double,location:URI): void
1
1
<<Thread>>
TaskSearchThread
1 1
<<LocalAspect>>
RAintra
1 1
<<LocalAspect>>
RAlocal

Figure 8.3. A detail presentation of the main class: SilbaBase.

Worker Agent

The worker agent (WA) extends SilbaBase and implements the NotificationListener interface of MozartSpaces. It also implements this interface's handleNotification method which fires every time a task got written to the load space. At startup of the task, the following arguments also must be specified:

- the maximum number of parallel threads for task computation allowed
- values for TP2 and TP3.

Upon notification, the WA checks whether the task is suitable for this WA. If a task matches, the WA first checks whether the task should be transferred, i.e., rescheduled immediately. This decision is based on the TP2 and TP3 threshold values, the worker agent speed (waSpeed) and the link speed (linkSpeed) as well as a transmission delay (td). The WA speed is given in Byte/ms as is the link speed. The complexity of a task is given in Bytes. If the number of tasks currently being processed by this WA is above the TP3 value, the Transfer Policy decides what to do. So, if complexity/waSpeed < complexity/linkSpeed + td, then the TP3 value is increased and the task does not get rescheduled. Otherwise the task is rejected and rescheduled and the TP3 stays the same. This should prevent an overall slow down of the make span if re-scheduling the task would take more time than computing it at the same WA. If a WAThread ends (i.e. a task is done), a task is taken from the waitingTaskList to the runningTaskList and thus starting a new WAThread. A WAThread is a thread that takes a task, processes it, writes the result into the answer container and measures how fast it was doing this. This information is needed for the Transfer Policy. The information whether a WA is underloaded/ok-loaded/overloaded (UL/OK/OL) is regularly written to the load space, together with more load information.

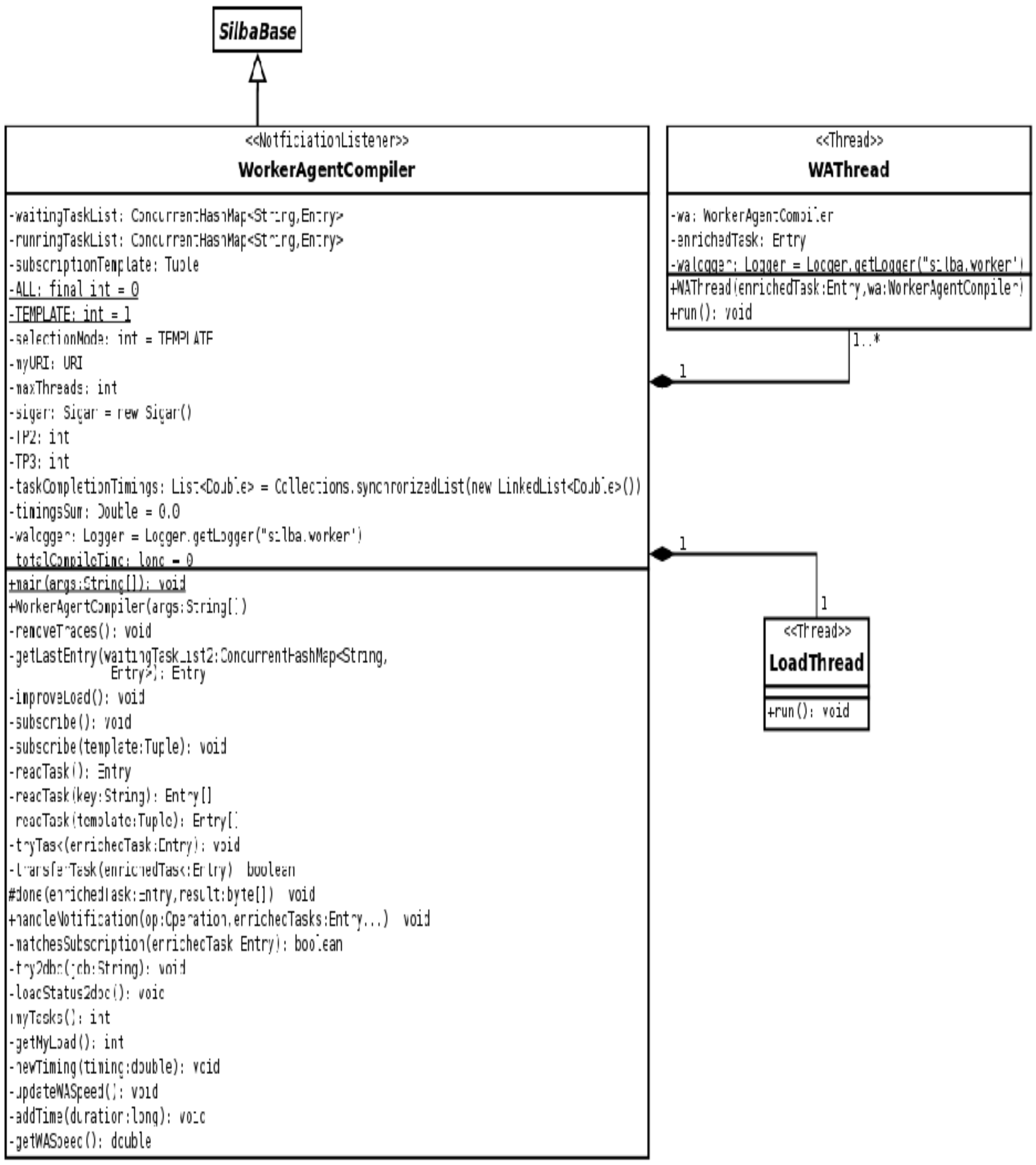

SilbaBase
<<NotficiationListener>>
WorkerAgentCompiler
-waitingTaskList: ConcurrentHashMap<String,Entry>
-runningTaskList: ConcurrentHashMap<String,Entry>
-subscriptionTemplate: Tuple
-ALL: final int = 0
-TEMPLATE: int = 1
-selectionMode: int = TEMPLATE
-myURI: URI
-maxThreads: int
-sigar: Sigar = new Sigar()
-TP2: int
-TP3: int
-taskCompletionTimings: List<Double> = Collections.synchronizedList(new LinkedList<Double>())
-timingsSum: Double = 0.0
-walogger: Logger = Logger.getLogger("silba.worker")
-totalCompileTime: long = 0
+main(args:String[]): void
+WorkerAgentCompiler(args:String[])
-removeTraces(): void
-getLastEntry(waitingTaskList2:ConcurrentHashMap<String, Entry>): Entry
-improveLoad(): void
-subscribe(): void
-subscribe(template:Tuple): void
-readTask(): Entry
-readTask(key:String): Entry[]
-readTask(template:Tuple): Entry[]
-tryTask(enrichedTask:Entry): void
-transferTask(enrichedTask:Entry): boolean
#done(enrichedTask:Entry,result:byte[]): void
+handleNotification(op:Operation,enrichedTasks:Entry...): void
-matchesSubscription(enrichedTask:Entry): boolean
-try2db(job:String): void
-loadStatus2db(): void
-myTasks(): int
-getMyLoad(): int
-newTiming(timing:double): void
-updateWASpeed(): void
-addTime(duration:long): void
-getWASpeed(): double
<<Thread>>
WAThread
-wa: WorkerAgentCompiler
-enrichedTask: Entry
-walogger: Logger = Logger.getLogger("silba.worker")
+WAThread(enrichedTask:Entry,wa:WorkerAgentCompiler)
+run(): void
1..*
1
1
1
<<Thread>>
LoadThread
+run(): void

Figure 8.4. A detail presentation of the main class: WorkerAgent

Routing Agent

The routing agent local is a MozartSpaces preWrite LocalAspect on the task container. This means every time some program attempts to write a new task into the load space, the RAlocal will fire. The task gets enriched with additional information (e.g., an assignment to a specific suitable worker agent, a more general worker agent role, a time stamp). A worker agent gets notified after the RAlocal added this information. A suitable WA is found by retrieving the worker agents' load information.

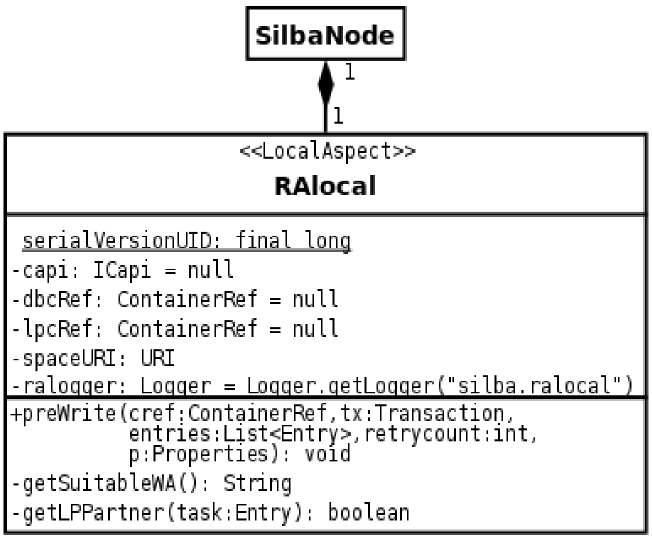

SilbaNode
1
1
<<LocalAspect>>
RAlocal
serialVersionUID: final long
-capi: ICapi = null
-dbcRef: ContainerRef = null
-lpcRef: ContainerRef = null
-spaceURI: URI
-ralogger: Logger = Logger.getLogger("silba.ralocal")
+preWrite(cref:ContainerRef,tx:Transaction, entries:List<Entry>,retrycount:int, p:Properties): void
-getSuitableWA(): String
-getLPPartner(task:Entry): boolean

Figure 8.5. A detail presentation of the main class: SilbaNode.

Appendix B

Appendix B contains the PlusCal algorithms of both architectures incl. their translations into TLA+ specification (first, Lookup module, second SILBA module).

```
--------------------- MODULE Lookup -------------------
  EXTENDS Naturals, TLC, Sequences
  \* M - the number of swarms
  \* N - the number of nodes

  CONSTANTS M, N

(* --algorithm Lookup {
\* msgC the array of channels for messages, spaces are simu-
lated as flexible channels, i.e., msgC[1],...,msgC[N] simu-
late swarm spaces, msgC[0] simulates the answer space
 variables msgC = [im \in 0 .. N |-> <<>>];
 define {
   clientNode == 0
   fromNode == 1
   currPath == 2
   pathPos == 3
   searchStr == 4
   swarmType == 5
   searchStatus == 6
   }

 macro WriteFIFO (m , chan) { chan := Append(chan, m ) }
 macro TakeFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan);
                              chan := Tail(chan)}
 macro ReadFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan)}
 macro WriteKEY (m , chan, key) { chan := [ikey \in 1 .. (IF
 Len(chan) < key THEN key ELSE Len(chan)) |-> IF ikey = key
 THEN m ELSE IF ikey <= Len(chan) THEN chan[ikey] ELSE <<>>]}

 macro TakeKEY  (v , chan, key) { await Len(chan) >= key /\
 chan[key] /= <<>>; v:=chan[key];
 chan := [ikey \in 1 .. Len(chan) |-> IF ikey = key THEN <<>>
 ELSE chan[ikey]]}
```

```
macro ReadKEY  (v , chan, key) { await Len(chan) >= key /\
chan[key] /= <<>>; v := chan[key]};

process (Client = 0)
\* msg - the message with the request
 variables msg = <<>>; initialPath = [ip \in 1 .. N |-> 0];
i; j; {
  \* send M messages, i.e., requests for searching
  l1 : i := 1;
  l2 : while (i <= M) {
     \* nondeterministically select some node j
      with (rndNode \in 1 .. N) {j := rndNode;};
      \* the message goes to node j
      WriteFIFO (<<clientNode, initialPath, 0, "searchStr",
                 "F", FALSE>>, msgC[j]);
      i:=i+1;
      };
 \* wait for processing the request
 l3 : while (i > 1) {
     \* take the message with the processed request msg from
     its channel, i.e., from the "answer space" msgC[0]
      TakeFIFO (msg, msgC[clientNode]);
      i := i-1;
      };
 \* assert that there is nothing left in channels, i.e., that
  the number of sent messages minus the number of received
  messages equals to 0
 l5 : while (i <= N) {
     assert (Len(msgC[i]) = 0);
     i:=i+1;
     };
 assert (Len(msgC[clientNode]) = 0);
}

process (Swarm \in 1 .. N)
variables msg = <<>>; nextNode; newPos; newType; newStatus;
i; iNodes {
 l1 : while (TRUE) {
      either skip;
      or {
     \* accept the message with request msg from its channel
       msgC[self]
      TakeFIFO (msg, msgC[self]);
      \* processing ...
      \* if the type is forward
      if (msg[swarmType] = "F") {
      \* the path lenght is increased by one
      newPos := msg[pathPos]+1;
      \* newStatus simulates a search in a content space
   with (rndFound \in {TRUE, FALSE}) {newStatus := rndFound};
      \* if newStatus is true (found) or the path length
      equals to N
      if(newStatus \/ newPos = N) {
      \* the type becomes backward
      newType := "B"
       };
      else {newType := "F"}
       };
      else {
      \* in "backward" case, the status of a search does not
      change and the current path length decreases by one
      newStatus := msg[searchStatus];
      newPos := msg[pathPos]-1;
      newType := "B";
      };
      \* selection of the next node
      if (newType = "F") {
      \* from set 1 .. N, exclude nodes that are in the path
      iNodes := 1 .. N \ {self};
      i := 1;
   l2 : while (i<newPos) {
      iNodes := iNodes \ {msg[currPath][i]};
      i := i+1;
       };
      \* from the rest of them, randomly (nondeterministi-
      cally) select one, i.e., this simulates the search and
      selection in the routing space
      with (rndNext \in iNodes) {nextNode := rndNext};
         };
       else {
      \* for the "backward" type, the value of a previous po-
      sition in the path refers to the previos node or if it
      is in the first position, the processed message is sent
      (routed back) to the client
      if (newPos=1) {
      nextNode := clientNode;
       };
      else {
      nextNode := msg[currPath][newPos-1];
          };
       };
      \* the message is processed
   l3 : msg[fromNode] := self || msg[currPath][newPos] :=
       self || msg[pathPos] := newPos || msg[swarmType] :=
       newType || msg[searchStatus] := newStatus;
       WriteFIFO(msg, msgC[nextNode]);
        };
      };
 }
}
*)

\* BEGIN TRANSLATION
\* Label l1 of process Client at line 43 col 9 changed to l1_
\* Label l2 of process Client at line 44 col 9 changed to l2_
\* Label l3 of process Client at line 54 col 9 changed to l3_
\* Process variable msg of process Client at line 40 col 12
changed to msg_
\* Process variable i of process Client at line 40 col 61
changed to i_
CONSTANT defaultInitValue
VARIABLES msgC, pc
 (* define statement *)
clientNode == 0
fromNode == 1
currPath == 2
pathPos == 3
searchStr == 4
swarmType == 5
searchStatus == 6

VARIABLES msg_, initialPath, i_, j, msg, nextNode, newPos,
newType, newStatus,
          i, iNodes

vars == << msgC, pc, msg_, initialPath, i_, j, msg, nextNode,
newPos, newType,
           newStatus, i, iNodes >>

ProcSet == {0} \cup (1 .. N)
```

```
Init == (* Global variables *)
        /\ msgC = [im \in 0 .. N |-> <<>>]
        (* Process Client *)
        /\ msg_ = <<>>
        /\ initialPath = [ip \in 1 .. N |-> 0]
        /\ i_ = defaultInitValue
        /\ j = defaultInitValue
        (* Process Swarm *)
        /\ msg = [self \in 1 .. N |-> <<>>]
        /\ nextNode = [self \in 1 .. N |-> defaultInitValue]
        /\ newPos = [self \in 1 .. N |-> defaultInitValue]
        /\ newType = [self \in 1 .. N |-> defaultInitValue]
        /\ newStatus = [self \in 1 .. N |-> defaultInitValue]
        /\ i = [self \in 1 .. N |-> defaultInitValue]
        /\ iNodes = [self \in 1 .. N |-> defaultInitValue]
        /\ pc = [self \in ProcSet |-> CASE self = 0 -> "l1_"
                                     [] self \in 1 .. N -> l1"]

l1_ == /\ pc[0] = "l1_"
       /\ i_' = 1
       /\ pc' = [pc EXCEPT ![0] = "l2_"]
       /\ UNCHANGED << msgC, msg_, initialPath, j, msg,
nextNode, newPos, newType, newStatus, i, iNodes >>

l2_ == /\ pc[0] = "l2_"
       /\ IF i_ <= M
             THEN /\ \E rndNode \in 1 .. N:
                       j' = rndNode
                  /\ msgC' = [msgC EXCEPT ![j'] = Ap-
pend((msgC[j']), (<<clientNode, initialPath, 0, "searchStr",
"F", FALSE>>) )]
                  /\ i_' = i_+1
                  /\ pc' = [pc EXCEPT ![0] = "l2_"]
             ELSE /\ pc' = [pc EXCEPT ![0] = "l3_"]
                  /\ UNCHANGED << msgC, i_, j >>
       /\ UNCHANGED << msg_, initialPath, msg, nextNode, new-
Pos, newType, newStatus, i, iNodes >>

l3_ == /\ pc[0] = "l3_"
       /\ IF i_ > 1
             THEN /\ (msgC[clientNode]) /= <<>>
                  /\ msg_' = Head((msgC[clientNode]))
                  /\ msgC' = [msgC EXCEPT ![clientNode] =
Tail((msgC[clientNode]))]
```

```
                  /\ i_' = i_-1
                  /\ pc' = [pc EXCEPT ![0] = "l3_"]
             ELSE /\ pc' = [pc EXCEPT ![0] = "l5"]
                  /\ UNCHANGED << msgC, msg_, i_ >>
       /\ UNCHANGED << initialPath, j, msg, nextNode, newPos,
newType, newStatus, i, iNodes >>

l5 == /\ pc[0] = "l5"
      /\ IF i_ <= N
            THEN /\ Assert((Len(msgC[i_]) = 0),
                           )
                 /\ i_' = i_+1
                 /\ pc' = [pc EXCEPT ![0] = "l5"]
            ELSE /\ Assert((Len(msgC[clientNode]) = 0),
                           ")
                 /\ pc' = [pc EXCEPT ![0] = "Done"]
                 /\ UNCHANGED i_
      /\ UNCHANGED << msgC, msg_, initialPath, j, msg,
nextNode, newPos, newType, newStatus, i, iNodes >>

Client == l1_ \/ l2_ \/ l3_ \/ l5

l1(self) == /\ pc[self] = "l1"
            /\ \/ /\ TRUE
                  /\ pc' = [pc EXCEPT ![self] = "l1"]
                  /\ UNCHANGED <<msgC, msg, nextNode, newPos,
newType, newStatus, i, iNodes>>
               \/ /\ (msgC[self]) /= <<>>
                  /\ msg' = [msg EXCEPT ![self] =
Head((msgC[self]))]
                  /\ msgC' = [msgC EXCEPT ![self] =
Tail((msgC[self]))]
                  /\ IF msg'[self][swarmType] = "F"
THEN /\ newPos' = [newPos EXCEPT ![self] =
msg'[self][pathPos]+1] /\ \E rndFound \in {TRUE, FALSE}:
newStatus' = [newStatus EXCEPT ![self] = rndFound]
                 /\ IF newStatus'[self] \/ newPos'[self] = N
THEN /\ newType' = [newType EXCEPT ![self] = "B"]
ELSE /\ newType' = [newType EXCEPT ![self] = "F"]
ELSE /\ newStatus' = [newStatus EXCEPT ![self] =
msg'[self][searchStatus]] /\ newPos' = [newPos EXCEPT
![self] = msg'[self][pathPos]-1]
    /\ newType' = [newType EXCEPT ![self] = "B"]
    /\ IF newType'[self] = "F
```

```
THEN /\ iNodes' = [iNodes EXCEPT ![self] = 1 .. N \ {self}]
     /\ i' = [i EXCEPT ![self] = 1]
     /\ pc' = [pc EXCEPT ![self] = "l2"]
     /\ UNCHANGED nextNode
ELSE /\ IF newPos'[self]=1
THEN /\ nextNode' = [nextNode EXCEPT ![self] = clientNode]
ELSE /\ nextNode' = [nextNode EXCEPT
![self] = msg'[self][currPath][newPos'[self]-1]]
     /\ pc' = [pc EXCEPT ![self] = "l3"]
     /\ UNCHANGED << i, iNodes >>
     /\ UNCHANGED << msg_, initialPath, i_, j >>

l2(self) == /\ pc[self] = "l2"
            /\ IF i[self]<newPos[self]
       THEN /\ iNodes' = [iNodes EXCEPT
![self] = iNodes[self] \ {msg[self][currPath][i[self]]}]
            /\ i' = [i EXCEPT ![self] = i[self]+1]
            /\ pc' = [pc EXCEPT ![self] = "l2"]
            /\ UNCHANGED nextNode
        ELSE /\ \E rndNext \in iNodes[self]:
            nextNode' = [nextNode EXCEPT ![self] = rndNext]
            /\ pc' = [pc EXCEPT ![self] = "l3"]
            /\ UNCHANGED << i, iNodes >>
            /\ UNCHANGED << msgC, msg_, initialPath, i_, j,
msg, newPos, newType, newStatus >>

l3(self) == /\ pc[self] = "l3"
            /\ msg' = [msg EXCEPT ![self][fromNode] = self,
![self][currPath][newPos[self]] = self,
![self][pathPos] = newPos[self],
![self][swarmType] = newType[self],
![self][searchStatus] = newStatus[self]]
            /\ msgC' = [msgC EXCEPT ![nextNode[self]] = Ap-
pend((msgC[nextNode[self]]), msg'[self] )]
            /\ pc' = [pc EXCEPT ![self] = "l1"]
            /\ UNCHANGED << msg_, initialPath, i_, j,
nextNode, newPos, newType, newStatus, i, iNodes >>

Swarm(self) == l1(self) \/ l2(self) \/ l3(self)
Next == Client
           \/ (\E self \in 1 .. N: Swarm(self))
           \/ (* Disjunct to prevent deadlock on termination
*)
((\A self \in ProcSet: pc[self] = "Done") /\ UNCHANGED vars)
```

```
Spec == Init /\ [][Next]_vars
Termination == <>(\A self \in ProcSet: pc[self] = "Done")

\* END TRANSLATION
=============================================================
```

Listing 8.1. The PlusCal algorithm with TLA+ specification (Lookup).

```
---------------------- MODULE Silba -------------------
EXTENDS Naturals, TLC, Sequences
\* M - the number of requests (tasks)
\* N - the number of nodes

CONSTANTS M, N, T1, T2

(* --algorithm Silba {
\* msgC - the array of channels for messages, spaces are
simulated as flexible channels, i.e., msgC[1],...,msgC[N]
simulate load spaces, msgC[0] simulates the answer space
\* allocC - the array of channels that simulate allocation
spaces
\* nStatus – the array of nodes’ status

 variables msgC = [im \in 0 .. N |-> <<>>]; allocC = [il \in
 1 .. N |-> <<>>]; nStatus = [in \in 1 .. N |-> "UL"];
 define {
   clientNode == 0
   fromNode == 1
   reqID == 2
   partnerNode == 3
   }

 macro WriteFIFO (m , chan) { chan := Append(chan, m ) }
 macro TakeFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan);
                              chan := Tail(chan)}
 macro ReadFIFO  (v , chan) { await chan /= <<>>;
                              v := Head(chan)}
 macro WriteKEY (m , chan, key) { chan := [ikey \in 1 .. (IF
 Len(chan) < key THEN key ELSE Len(chan)) |-> IF ikey = key
 THEN m ELSE IF ikey <= Len(chan) THEN chan[ikey] ELSE <<>>]}

 macro TakeKEY  (v , chan, key) { await Len(chan) >= key /\
 chan[key] /= <<>>; v:=chan[key];
 chan := [ikey \in 1 .. Len(chan) |-> IF ikey = key THEN <<>>
 ELSE chan[ikey]]}

 macro ReadKEY  (v , chan, key) { await Len(chan) >= key /\
 chan[key] /= <<>>; v := chan[key]};

process (Client = 0)
\* msg - the message with the request
 variables msg = <<>>; i; j; {
   \* send M messages, i.e., requests
   l1 : i := 1;
   l2 : while (i <= M) {
   \* select randomly (nondeterministically) some node j
    with (rndNode \in 1 .. N) { j := rndNode;};
   \* the message goes to the msg channel of node j, i.e., to
    the "load space" of node j
    WriteFIFO (<<clientNode, i, clientNode>>, msgC[j]);
    i:=i+1;
     };
   \* wait for processing the request
   l3 : while (i > 1) {
   \* accept the message with the processed request msg from
   its channel msgC[0], i.e., msgC[0] refers to the answer
   space
    ReadKEY (msg, msgC[clientNode], i-1);
    i:=i-1;
    };
   \* assert that there is nothing left in channels, i.e.,
   that the number of sent messages minus the number of re-
   ceived messages equals to 0
   l4 : while (i <= N) {
        assert (Len(msgC[i]) = 0 /\ Len(allocC[i])=0);
        i:=i+1;
       };
   assert (Len(msgC[clientNode]) = M);
 }

process (WA \in 1 .. N)
variables msg = <<>>; {
   l1 : while (nStatus[self] /= "OL") {
        either skip
        or {
       \* accept the message with the request msg from its
       channel msgC[self], i.e., "load space"
       TakeFIFO (msg, msgC[self]);
       \* processing ...
       skip; \* "execute" some job
     l2 : msg[fromNode] := self;
       WriteKEY(msg, msgC[clientNode], msg[reqID]);
       };
  l3 : if (Len(msgC[self]) < T1) {
        nStatus[self] := "UL"
        }
        else {
        if (Len(msgC[self]) < T2) nStatus[self] := "OK";
        else nStatus[self] := "OL"
         }
      };
 }

process (Arbiter \in 1 .. N)
variables msg = <<>>; {
  l1 : while (nStatus[self] = "OL") {
       either skip
       or {
       \* accept the message with the request msg from its
       channel msgC[self], i.e., "load space"
       TakeFIFO (msg, msgC[self]);
      \* send the request to allocC, i.e., "allocation space"
     l2 : msg[fromNode] := self;
           WriteFIFO(msg, allocC[self]);
        };
      };
 }

process (RA \in 1 .. N)
variables msg = <<>>; ack; i; pNodes; pNode; {
  l1 : while (TRUE) {
       either {
        if (nStatus[i]="UL") {
        i := 1;
        pNodes := 1 .. N;
       l2 : pNodes := pNodes \ {self};
       l3 : while (i<=N) {
            if (nStatus[i]/="OL") pNodes := pNodes \ {i};
            i := i+1;
             };
            with (rndNode \in pNodes) {pNode := rndNode};
            WriteFIFO(<<self, 0, pNode>>, allocC[self]);
            };
         };
       or {
       \* accept the message with the request msg from its
        channel allocC[self], i.e., "allocation space"
          ReadFIFO (msg, allocC[self]);
          if (msg[partnerNode] = clientNode) {
          \* find UL or OK node
          i := 1;
          pNodes := 1 .. N;
         l4 : pNodes := pNodes \ {self};
         l5 : while (i<=N) {
            if (nStatus[i]="OL") pNodes := pNodes \ {i};
            i := i+1;
              };
            with (rndNode \in pNodes) {pNode := rndNode};
            allocC[self][1][partnerNode] := pNode;
             };
          };
      };
 }

process (OUTag \in 1 .. N)
variables msg = <<>>; pNode; {
  l1 : while (TRUE) {
       either skip
       or {
       \* accept the message from the channel
        ReadFIFO (msg, allocC[self]);
        if (msg[partnerNode] /= clientNode /\ msg[reqID]>0) {
        \* take the message from allocC and write it to the
        "load space" of pNode
       l2 : TakeFIFO (msg, allocC[self]);
           pNode := msg[partnerNode];
       l3 : msg[fromNode] := self || msg[partnerNode] := cli-
entNode;
           WriteFIFO(msg, msgC[pNode]);
            };
        };
      };
 }

process (INag \in 1 .. N)
variables msg = <<>>; pNode; {
  l1 : while (TRUE) {
       either skip
       or {
       \* accept the message from the channel
        ReadFIFO (msg, allocC[self]);
        if (msg[partnerNode] /= clientNode /\ msg[reqID]=0) {
        \* take the message from allocC and write it to the
        "load space" of pNode
      l2 : TakeFIFO (msg, allocC[self]);
           pNode := msg[partnerNode];
      l3 : TakeFIFO(msg, msgC[pNode]);
     l4 : msg[fromNode] := self || msg[partnerNode] := cli-
entNode;
         WriteFIFO(msg, msgC[self]);
            };
        };
     };
 }

}
*)

\* BEGIN TRANSLATION
\* Label l1 of process Client at line 42 col 10 changed to
l1_
\* Label l2 of process Client at line 43 col 10 changed to
l2_
\* Label l3 of process Client at line 52 col 10 changed to
l3_
\* Label l4 of process Client at line 59 col 10 changed to
l4_
\* Label l1 of process WA at line 69 col 9 changed to l1_W
\* Label l2 of process WA at line 77 col 16 changed to l2_W
\* Label l3 of process WA at line 80 col 11 changed to l3_W
\* Label l1 of process Arbiter at line 92 col 9 changed to
l1_A
\* Label l2 of process Arbiter at line 98 col 16 changed to
l2_A
```

```
\* Label l1 of process RA at line 106 col 9 changed to l1_R
\* Label l2 of process RA at line 111 col 18 changed to l2_R
\* Label l3 of process RA at line 112 col 18 changed to l3_R
\* Label l4 of process RA at line 128 col 18 changed to l4_R
\* Label l1 of process OUTag at line 142 col 9 changed to
l1_O
\* Label l2 of process OUTag at line 23 col 31 changed to
l2_O
\* Label l3 of process OUTag at line 151 col 18 changed to
l3_O
\* Process variable msg of process Client at line 39 col 12
changed to msg_
\* Process variable i of process Client at line 39 col 24
changed to i_
\* Process variable msg of process WA at line 68 col 12
changed to msg_W
\* Process variable msg of process Arbiter at line 91 col 12
changed to msg_A
\* Process variable msg of process RA at line 105 col 12
changed to msg_R
\* Process variable pNode of process RA at line 105 col 40
changed to pNode_
\* Process variable msg of process OUTag at line 141 col 12
changed to msg_O
\* Process variable pNode of process OUTag at line 141 col 24
changed to pNode_O
CONSTANT defaultInitValue
VARIABLES msgC, allocC, nStatus, pc

(* define statement *)
clientNode == 0
fromNode == 1
reqID == 2
partnerNode == 3

VARIABLES msg_, i_, j, msg_W, msg_A, msg_R, ack, i, pNodes,
pNode_, msg_O,
          pNode_O, msg, pNode

vars == << msgC, allocC, nStatus, pc, msg_, i_, j, msg_W,
msg_A, msg_R, ack,
           i, pNodes, pNode_, msg_O, pNode_O, msg, pNode >>
```

```
ProcSet == {0} \cup (1 .. N) \cup (1 .. N) \cup (1 .. N) \cup
(1 .. N) \cup (1 .. N)

Init == (* Global variables *)
        /\ msgC = [im \in 0 .. N |-> <<>>]
        /\ allocC = [il \in 1 .. N |-> <<>>]
        /\ nStatus = [in \in 1 .. N |-> "UL"]
        (* Process Client *)
        /\ msg_ = <<>>
        /\ i_ = defaultInitValue
        /\ j = defaultInitValue
        (* Process WA *)
        /\ msg_W = [self \in 1 .. N |-> <<>>]
        (* Process Arbiter *)
        /\ msg_A = [self \in 1 .. N |-> <<>>]
        (* Process RA *)
        /\ msg_R = [self \in 1 .. N |-> <<>>]
        /\ ack = [self \in 1 .. N |-> defaultInitValue]
        /\ i = [self \in 1 .. N |-> defaultInitValue]
        /\ pNodes = [self \in 1 .. N |-> defaultInitValue]
        /\ pNode_ = [self \in 1 .. N |-> defaultInitValue]
        (* Process OUTag *)
        /\ msg_O = [self \in 1 .. N |-> <<>>]
        /\ pNode_O = [self \in 1 .. N |-> defaultInitValue]
        (* Process INag *)
        /\ msg = [self \in 1 .. N |-> <<>>]
        /\ pNode = [self \in 1 .. N |-> defaultInitValue]
        /\ pc = [self \in ProcSet |-> CASE self = 0 -> "l1_"
                                 [] self \in 1 .. N -> "l1_W"
                                 [] self \in 1 .. N -> "l1_A"
                                 [] self \in 1 .. N -> "l1_R"
                                 [] self \in 1 .. N -> "l1_O"
                                 [] self \in 1 .. N -> "l1"]

l1_ == /\ pc[0] = "l1_"
       /\ i_' = 1
       /\ pc' = [pc EXCEPT ![0] = "l2_"]
       /\ UNCHANGED << msgC, allocC, nStatus, msg_, j, msg_W,
msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg,
pNode >>

l2_ == /\ pc[0] = "l2_"
       /\ IF i_ <= M
          THEN /\ \E rndNode \in 1 .. N:
```

```
                     j' = rndNode
                  /\ msgC' = [msgC EXCEPT ![j'] = Ap-
pend((msgC[j']), (<<clientNode, i_, clientNode>>) )]
                  /\ i_' = i_+1
                  /\ pc' = [pc EXCEPT ![0] = "l2_"]
             ELSE /\ pc' = [pc EXCEPT ![0] = "l3_"]
                  /\ UNCHANGED << msgC, i_, j >>
       /\ UNCHANGED << allocC, nStatus, msg_, msg_W, msg_A,
msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg, pNode >>

l3_ == /\ pc[0] = "l3_"
       /\ IF i_ > 1
          THEN /\ Len((msgC[clientNode])) >= (i_-1) /\
(msgC[clientNode])[(i_-1)] /= <<>>
                  /\ msg_' = (msgC[clientNode])[(i_-1)]
                  /\ i_' = i_-1
                  /\ pc' = [pc EXCEPT ![0] = "l3_"]
             ELSE /\ pc' = [pc EXCEPT ![0] = "l4_"]
                  /\ UNCHANGED << msg_, i_ >>
       /\ UNCHANGED << msgC, allocC, nStatus, j, msg_W,
msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg,
pNode >>

l4_ == /\ pc[0] = "l4_"
       /\ IF i_ <= N
             THEN /\ Assert((Len(msgC[i_]) = 0 /\
Len(allocC[i_])=0),
                              )
                  /\ i_' = i_+1
                  /\ pc' = [pc EXCEPT ![0] = "l4_"]
             ELSE /\ Assert((Len(msgC[clientNode]) = M),
                              )
                  /\ pc' = [pc EXCEPT ![0] = "Done"]
                  /\ UNCHANGED i_
       /\ UNCHANGED << msgC, allocC, nStatus, msg_, j, msg_W,
msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg,
pNode >>

Client == l1_ \/ l2_ \/ l3_ \/ l4_

l1_W(self) == /\ pc[self] = "l1_W"
              /\ IF nStatus[self] /= "OL"
                 THEN /\ \/ /\ TRUE
              /\ pc' = [pc EXCEPT ![self] = "l3_W"]
              /\ UNCHANGED <<msgC, msg_W>>
              \/ /\ (msgC[self]) /= <<>>
      /\ msg_W' = [msg_W EXCEPT ![self] = Head((msgC[self]))]
      /\ msgC' = [msgC EXCEPT ![self] = Tail((msgC[self]))]
              /\ TRUE
              /\ pc' = [pc EXCEPT ![self] = "l2_W"]
              ELSE /\ pc' = [pc EXCEPT ![self] = "Done"]
              /\ UNCHANGED << msgC, msg_W >>
              /\ UNCHANGED << allocC, nStatus, msg_, i_, j,
msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg,
pNode >>

l3_W(self) == /\ pc[self] = "l3_W"
              /\ IF Len(msgC[self]) < T1
          THEN /\ nStatus' = [nStatus EXCEPT ![self] = "UL"]
          ELSE /\ IF Len(msgC[self]) < T2
          THEN /\ nStatus' = [nStatus EXCEPT ![self] = "OK"]
          ELSE /\ nStatus' = [nStatus EXCEPT ![self] = "OL"]
               /\ pc' = [pc EXCEPT ![self] = "l1_W"]
               /\ UNCHANGED << msgC, allocC, msg_, i_, j,
msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O,
msg, pNode >>

l2_W(self) == /\ pc[self] = "l2_W"
         /\ msg_W' = [msg_W EXCEPT ![self][fromNode] = self]
         /\ msgC' = [msgC EXCEPT ![clientNode] = [ikey \in 1
.. (IF Len((msgC[clientNode])) < (msg_W'[self][reqID]) THEN
(msg_W'[self][reqID]) ELSE Len((msgC[clientNode]))) |-> IF
ikey = (msg_W'[self][reqID]) THEN msg_W'[self] ELSE
IF ikey <= Len((msgC[clientNode])) THEN
(msgC[clientNode])[ikey] ELSE <<>>]]
              /\ pc' = [pc EXCEPT ![self] = "l3_W"]
              /\ UNCHANGED << allocC, nStatus, msg_, i_, j,
msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg,
pNode >>

WA(self) == l1_W(self) \/ l3_W(self) \/ l2_W(self)

l1_A(self) == /\ pc[self] = "l1_A"
              /\ IF nStatus[self] = "OL"
              THEN /\ \/ /\ TRUE
               /\ pc' = [pc EXCEPT ![self] = "l1_A"]
               /\ UNCHANGED <<msgC, msg_A>>
               \/ /\ (msgC[self]) /= <<>>
```

```
     /\ msg_A' = [msg_A EXCEPT ![self] = Head((msgC[self]))]
     /\ msgC' = [msgC EXCEPT ![self] = Tail((msgC[self]))]
     /\ pc' = [pc EXCEPT ![self] = "l2_A"]
              ELSE /\ pc' = [pc EXCEPT ![self] = "Done"]
              /\ UNCHANGED << msgC, msg_A >>
              /\ UNCHANGED << allocC, nStatus, msg_, i_, j,
msg_W, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg,
pNode >>

l2_A(self) == /\ pc[self] = "l2_A"
         /\ msg_A' = [msg_A EXCEPT ![self][fromNode] = self]
         /\ allocC' = [allocC EXCEPT ![self] = Ap-
pend((allocC[self]), msg_A'[self] )]
         /\ pc' = [pc EXCEPT ![self] = "l1_A"]
         /\ UNCHANGED << msgC, nStatus, msg_, i_, j, msg_W,
msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O, msg, pNode >>

Arbiter(self) == l1_A(self) \/ l2_A(self)

l1_R(self) == /\ pc[self] = "l1_R"
              /\ \/ /\ IF nStatus[i[self]]="UL"
                 THEN /\ i' = [i EXCEPT ![self] = 1]
               /\ pNodes' = [pNodes EXCEPT ![self] = 1 .. N]
               /\ pc' = [pc EXCEPT ![self] = "l2_R"]
               ELSE /\ pc' = [pc EXCEPT ![self] = "l1_R"]
               /\ UNCHANGED << i, pNodes >>
               /\ UNCHANGED msg_R
               \/ /\ (allocC[self]) /= <<>>
/\ msg_R' = [msg_R EXCEPT ![self] = Head((allocC[self]))]
              /\ IF msg_R'[self][partnerNode] = clientNode
               THEN /\ i' = [i EXCEPT ![self] = 1]
              /\ pNodes' = [pNodes EXCEPT ![self] = 1 .. N]
              /\ pc' = [pc EXCEPT ![self] = "l4_R"]
              ELSE /\ pc' = [pc EXCEPT ![self] = "l1_R"]
              /\ UNCHANGED << i, pNodes >>
              /\ UNCHANGED << msgC, allocC, nStatus, msg_,
i_, j, msg_W, msg_A, ack, pNode_, msg_O, pNode_O, msg, pNode
>>

l2_R(self) == /\ pc[self] = "l2_R"
 /\ pNodes' = [pNodes EXCEPT ![self] = pNodes[self] \ {self}]
              /\ pc' = [pc EXCEPT ![self] = "l3_R"]
              /\ UNCHANGED << msgC, allocC, nStatus, msg_,
i_, j, msg_W, msg_A, msg_R, ack, i, pNode_, msg_O, pNode_O,
msg, pNode >>

l3_R(self) == /\ pc[self] = "l3_R"
              /\ IF i[self]<=N
            THEN /\ IF nStatus[i[self]]/="OL"
            THEN /\ pNodes' = [pNodes EXCEPT
            ![self] = pNodes[self] \ {i[self]}]
             ELSE /\ TRUE
             /\ UNCHANGED pNodes
             /\ i' = [i EXCEPT ![self] = i[self]+1]
             /\ pc' = [pc EXCEPT ![self] = "l3_R"]
             /\ UNCHANGED << allocC, pNode_ >>
             ELSE /\ \E rndNode \in pNodes[self]:
             pNode_' = [pNode_ EXCEPT ![self] = rndNode]
             /\ allocC' = [allocC EXCEPT ![self] = Ap-
pend((allocC[self]), (<<self, 0, pNode_'[self]>>) )]
             /\ pc' = [pc EXCEPT ![self] = "l1_R"]
             /\ UNCHANGED << i, pNodes >>
             /\ UNCHANGED << msgC, nStatus, msg_, i_, j,
msg_W, msg_A, msg_R, ack, msg_O, pNode_O, msg, pNode >>

l4_R(self) == /\ pc[self] = "l4_R"
 /\ pNodes' = [pNodes EXCEPT ![self] = pNodes[self] \ {self}]
              /\ pc' = [pc EXCEPT ![self] = "l5"]
              /\ UNCHANGED << msgC, allocC, nStatus, msg_,
i_, j, msg_W, msg_A, msg_R, ack, i, pNode_, msg_O, pNode_O,
msg, pNode >>

l5(self) == /\ pc[self] = "l5"
            /\ IF i[self]<=N
           THEN /\ IF nStatus[i[self]]="OL"
           THEN /\ pNodes' = [pNodes EXCEPT
           ![self] = pNodes[self] \ {i[self]}]
           ELSE /\ TRUE
           /\ UNCHANGED pNodes
           /\ i' = [i EXCEPT ![self] = i[self]+1]
           /\ pc' = [pc EXCEPT ![self] = "l5"]
           /\ UNCHANGED << allocC, pNode_ >>
           ELSE /\ \E rndNode \in pNodes[self]:
           pNode_' = [pNode_ EXCEPT ![self] = rndNode]
           /\ allocC' = [allocC EXCEPT
          ![self][1][partnerNode] = pNode_'[self]]
```

```
           /\ pc' = [pc EXCEPT ![self] = "l1_R"]
           /\ UNCHANGED << i, pNodes >>
           /\ UNCHANGED << msgC, nStatus, msg_, i_, j, msg_W,
msg_A, msg_R, ack, msg_O, pNode_O, msg, pNode >>

RA(self) == l1_R(self) \/ l2_R(self) \/ l3_R(self) \/
l4_R(self) \/ l5(self)

l1_O(self) == /\ pc[self] = "l1_O"
              /\ \/ /\ TRUE
                    /\ pc' = [pc EXCEPT ![self] = "l1_O"]
                    /\ UNCHANGED msg_O
                 \/ /\ (allocC[self]) /= <<>>
                    /\ msg_O' = [msg_O EXCEPT
                       ![self] = Head((allocC[self]))]
                    /\ IF msg_O'[self][partnerNode] /= cli-
entNode /\ msg_O'[self][reqID]>0
                    THEN /\ pc' = [pc EXCEPT ![self] = "l2_O"]
                    ELSE /\ pc' = [pc EXCEPT ![self] = "l1_O"]
              /\ UNCHANGED << msgC, allocC, nStatus, msg_,
i_, j, msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, pNode_O,
msg, pNode >>

l2_O(self) == /\ pc[self] = "l2_O"
              /\ (allocC[self]) /= <<>>
/\ msg_O' = [msg_O EXCEPT ![self] = Head((allocC[self]))]
/\ allocC' = [allocC EXCEPT ![self] = Tail((allocC[self]))]
/\ pNode_O' = [pNode_O EXCEPT ![self] =
msg_O'[self][partnerNode]]
              /\ pc' = [pc EXCEPT ![self] = "l3_O"]
              /\ UNCHANGED << msgC, nStatus, msg_, i_, j,
msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, msg, pNode >>

l3_O(self) == /\ pc[self] = "l3_O"
/\ msg_O' = [msg_O EXCEPT ![self][fromNode] = self,
             ![self][partnerNode] = clientNode]
/\ msgC' = [msgC EXCEPT ![pNode_O[self]] = Ap-
pend((msgC[pNode_O[self]]), msg_O'[self] )]
             /\ pc' = [pc EXCEPT ![self] = "l1_O"]
             /\ UNCHANGED << allocC, nStatus, msg_, i_, j,
msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, pNode_O, msg,
                              pNode >>

OUTag(self) == l1_O(self) \/ l2_O(self) \/ l3_O(self)
```

```
l1(self) == /\ pc[self] = "l1"
            /\ \/ /\ TRUE
                  /\ pc' = [pc EXCEPT ![self] = "l1"]
                  /\ UNCHANGED msg
               \/ /\ (allocC[self]) /= <<>>
/\ msg' = [msg EXCEPT ![self] = Head((allocC[self]))]
/\ IF msg'[self][partnerNode] /= clientNode /\
msg'[self][reqID]=0
             THEN /\ pc' = [pc EXCEPT ![self] = "l2"]
             ELSE /\ pc' = [pc EXCEPT ![self] = "l1"]
            /\ UNCHANGED << msgC, allocC, nStatus, msg_, i_,
j, msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, msg_O,
pNode_O, pNode >>

l2(self) == /\ pc[self] = "l2"
            /\ (allocC[self]) /= <<>>
/\ msg' = [msg EXCEPT ![self] = Head((allocC[self]))]
/\ allocC' = [allocC EXCEPT ![self] = Tail((allocC[self]))]
/\ pNode' = [pNode EXCEPT ![self] = msg'[self][partnerNode]]
           /\ pc' = [pc EXCEPT ![self] = "l3"]
           /\ UNCHANGED << msgC, nStatus, msg_, i_, j, msg_W,
msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O >>

l3(self) == /\ pc[self] = "l3"
            /\ (msgC[pNode[self]]) /= <<>>
/\ msg' = [msg EXCEPT ![self] = Head((msgC[pNode[self]]))]
/\ msgC' = [msgC EXCEPT ![pNode[self]] =
Tail((msgC[pNode[self]]))]
            /\ pc' = [pc EXCEPT ![self] = "l4"]
            /\ UNCHANGED << allocC, nStatus, msg_, i_, j,
msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O,
pNode >>

l4(self) == /\ pc[self] = "l4"
/\ msg' = [msg EXCEPT ![self][fromNode] = self,
                     ![self][partnerNode] = clientNode]
/\ msgC' = [msgC EXCEPT ![self] = Append((msgC[self]),
msg'[self] )]
            /\ pc' = [pc EXCEPT ![self] = "l1"]
            /\ UNCHANGED << allocC, nStatus, msg_, i_, j,
msg_W, msg_A, msg_R, ack, i, pNodes, pNode_, msg_O, pNode_O,
pNode >>
```

```
INag(self) == l1(self) \/ l2(self) \/ l3(self) \/ l4(self)

Next == Client
           \/ (\E self \in 1 .. N: WA(self))
           \/ (\E self \in 1 .. N: Arbiter(self))
           \/ (\E self \in 1 .. N: RA(self))
           \/ (\E self \in 1 .. N: OUTag(self))
           \/ (\E self \in 1 .. N: INag(self))
           \/ (* Disjunct to prevent deadlock on termination
*)
              ((\A self \in ProcSet: pc[self] = "Done") /\
UNCHANGED vars)

Spec == Init /\ [][Next]_vars

Termination == <>(\A self \in ProcSet: pc[self] = "Done")

\* END TRANSLATION
=============================================================
```

Listing 8.2. The PlusCal algorithm with TLA+ specification (SILBA).

Appendix C

Appendix C contains just a very small portion of the huge amount of benchmarking results in order to illustrate fine tuning of parameters. For example, tables 8.1-8.5 contain the results needed for only one figure 6.1. The result (best tour length in ms) represents an average of 10 test-runs, (i.e., to obtain the result – the best tour length – 10 test-runs were performed with the same values of *α, β, ρ*, and then the average is calculated).

Tables 8.6.-8.10 have the same meaning and the final result of these tables (also each row is performed 10 times) is presented on Fig. 6.3.

The next part of tables (8.11-8.14) represent already summarized one part of the results from the second scenario that refers to the extended SILBA framework. Also, to obtain the result of only one row, each combination is done 10 times (explained in Chapter 6) and the average is calculated.

Table 8.1 Variation of parameters in random/MMAS (40 containers)

best tour length (in ms)	α	β	ρ
222	0,00	2,00	0,50
175	0,00	2,00	0,70
109	0,00	2,00	0,90
157	0,00	3,00	0,50
371	0,00	3,00	0,70
293	0,00	3,00	0,90
380	0,00	4,00	0,50
137	0,00	4,00	0,70
170	0,00	4,00	0,90
92	0,00	5,00	0,50
160	0,00	5,00	0,70
258	0,00	5,00	0,90
261	0,50	2,00	0,50
179	0,50	2,00	0,70
353	0,50	2,00	0,90
285	0,50	3,00	0,50
196	0,50	3,00	0,70
327	0,50	3,00	0,90

142	0,50	4,00	0,50
276	0,50	4,00	0,70
131	0,50	4,00	0,90
110	0,50	5,00	0,50
137	0,50	5,00	0,70
296	0,50	5,00	0,90
278	1,00	2,00	0,50
189	1,00	2,00	0,70
177	1,00	2,00	0,90
127	1,00	3,00	0,50
180	1,00	3,00	0,70
201	1,00	3,00	0,90
200	1,00	4,00	0,50
230	1,00	4,00	0,70
265	1,00	4,00	0,90
172	1,00	5,00	0,50
189	1,00	5,00	0,70
147	1,00	5,00	0,90

Table 8.2. Variation of parameters in random/MMAS (80 containers)

best tour length (in ms)	α	β	ρ
138	0,00	2,00	0,50
342	0,00	2,00	0,70
813	0,00	2,00	0,90
247	0,00	3,00	0,50
716	0,00	3,00	0,70
378	0,00	3,00	0,90
299	0,00	4,00	0,50
830	0,00	4,00	0,70
844	0,00	4,00	0,90
128	0,00	5,00	0,50
160	0,00	5,00	0,70
510	0,00	5,00	0,90

147	0,50	2,00	0,50
886	0,50	2,00	0,70
285	0,50	2,00	0,90
192	0,50	3,00	0,50
293	0,50	3,00	0,70
324	0,50	3,00	0,90
271	0,50	4,00	0,50
675	0,50	4,00	0,70
177	0,50	4,00	0,90
143	0,50	5,00	0,50
168	0,50	5,00	0,70
338	0,50	5,00	0,90
838	1,00	2,00	0,50
200	1,00	2,00	0,70
881	1,00	2,00	0,90
150	1,00	3,00	0,50
750	1,00	3,00	0,70
791	1,00	3,00	0,90
719	1,00	4,00	0,50
332	1,00	4,00	0,70
160	1,00	4,00	0,90
685	1,00	5,00	0,50
288	1,00	5,00	0,70
173	1,00	5,00	0,90

Table 8.3. Variation of parameters in random/MMAS (120 containers)

best tour length (in ms)	α	β	ρ
304	0,00	2,00	0,50
217	0,00	2,00	0,70
362	0,00	2,00	0,90
309	0,00	3,00	0,50
204	0,00	3,00	0,70
607	0,00	3,00	0,90
397	0,00	4,00	0,50

150	0,00	4,00	0,70
697	0,00	4,00	0,90
163	0,00	5,00	0,50
254	0,00	5,00	0,70
572	0,00	5,00	0,90
703	0,50	2,00	0,50
391	0,50	2,00	0,70
1129	0,50	2,00	0,90
297	0,50	3,00	0,50
515	0,50	3,00	0,70
294	0,50	3,00	0,90
512	0,50	4,00	0,50
760	0,50	4,00	0,70
857	0,50	4,00	0,90
175	0,50	5,00	0,50
448	0,50	5,00	0,70
536	0,50	5,00	0,90
218	1,00	2,00	0,50
625	1,00	2,00	0,70
420	1,00	2,00	0,90
173	1,00	3,00	0,50
358	1,00	3,00	0,70
499	1,00	3,00	0,90
445	1,00	4,00	0,50
389	1,00	4,00	0,70
210	1,00	4,00	0,90
366	1,00	5,00	0,50
662	1,00	5,00	0,70
403	1,00	5,00	0,90

Table 8.4. Variation of parameters in random/MAS (160 containers)

best tour length (in ms)	α	β	ρ
504	0,00	2,00	0,50
456	0,00	2,00	0,70
744	0,00	2,00	0,90
489	0,00	3,00	0,50
677	0,00	3,00	0,70
953	0,00	3,00	0,90
1047	0,00	4,00	0,50
639	0,00	4,00	0,70
872	0,00	4,00	0,90
199	0,00	5,00	0,50
1389	0,00	5,00	0,70
1572	0,00	5,00	0,90
763	0,50	2,00	0,50
1492	0,50	2,00	0,70
652	0,50	2,00	0,90
886	0,50	3,00	0,50
923	0,50	3,00	0,70
622	0,50	3,00	0,90
398	0,50	4,00	0,50
860	0,50	4,00	0,70
758	0,50	4,00	0,90
208	0,50	5,00	0,50
884	0,50	5,00	0,70
653	0,50	5,00	0,90
812	1,00	2,00	0,50
456	1,00	2,00	0,70
675	1,00	2,00	0,90
205	1,00	3,00	0,50
599	1,00	3,00	0,70
701	1,00	3,00	0,90
1345	1,00	4,00	0,50
1501	1,00	4,00	0,70

782	1,00	4,00	0,90
924	1,00	5,00	0,50
1093	1,00	5,00	0,70
1500	1,00	5,00	0,90

Table 8.5. Variation of parameters in random/MMAS (200 containers)

best tour length (in ms)	α	β	ρ
715	0,00	2,00	0,50
712	0,00	2,00	0,70
502	0,00	2,00	0,90
1309	0,00	3,00	0,50
1250	0,00	3,00	0,70
775	0,00	3,00	0,90
859	0,00	4,00	0,50
924	0,00	4,00	0,70
1650	0,00	4,00	0,90
234	0,00	5,00	0,50
448	0,00	5,00	0,70
557	0,00	5,00	0,90
703	0,50	2,00	0,50
931	0,50	2,00	0,70
1653	0,50	2,00	0,90
927	0,50	3,00	0,50
889	0,50	3,00	0,70
673	0,50	3,00	0,90
837	0,50	4,00	0,50
1294	0,50	4,00	0,70
583	0,50	4,00	0,90
240	0,50	5,00	0,50
572	0,50	5,00	0,70
534	0,50	5,00	0,90
782	1,00	2,00	0,50
547	1,00	2,00	0,70

499	1,00	2,00	0,90
245	1,00	3,00	0,50
857	1,00	3,00	0,70
1175	1,00	3,00	0,90
927	1,00	4,00	0,50
366	1,00	4,00	0,70
483	1,00	4,00	0,90
1366	1,00	5,00	0,50
712	1,00	5,00	0,70
736	1,00	5,00	0,90

Table 8.6. Variation of parameters in random/AntNet (40 containers)

best tour length (in ms)	α	C_2
148	0,20	0,15
95	0,20	0,20
80	0,20	0,25
100	0,20	0,30
76	0,20	0,35
112	0,30	0,15
104	0,30	0,20
165	0,30	0,25
136	0,30	0,30
92	0,30	0,35
129	0,40	0,15
92	0,40	0,20
92	0,40	0,25
120	0,40	0,30
100	0,40	0,35
88	0,50	0,15
105	0,50	0,20
117	0,50	0,25
99	0,50	0,30
120	0,50	0,35

Table 8.7. Variation of parameters in random/AntNet (80 containers)

best tour length (in ms)	α	C_2
155	0,20	0,15
102	0,20	0,20
96	0,20	0,25
110	0,20	0,30
99	0,20	0,35
126	0,30	0,15
155	0,30	0,20
173	0,30	0,25
180	0,30	0,30
101	0,30	0,35
136	0,40	0,15
111	0,40	0,20
113	0,40	0,25
147	0,40	0,30
124	0,40	0,35
109	0,50	0,15
166	0,50	0,20
149	0,50	0,25
111	0,50	0,30
169	0,50	0,35

Table 8.8. Variation of parameters in random/AntNet (120 containers)

best tour length (in ms)	α	C_2
297	0,20	0,15
205	0,20	0,20
130	0,20	0,25
155	0,20	0,30
148	0,20	0,35
299	0,30	0,15
267	0,30	0,20

308	0,30	0,25
275	0,30	0,30
178	0,30	0,35
205	0,40	0,15
177	0,40	0,20
183	0,40	0,25
253	0,40	0,30
201	0,40	0,35
176	0,50	0,15
212	0,50	0,20
223	0,50	0,25
181	0,50	0,30
254	0,50	0,35

Table 8.9. Variation of parameters in random/AntNet (160 containers)

best tour length (in ms)	α	C_2
301	0,20	0,15
295	0,20	0,20
152	0,20	0,25
226	0,20	0,30
198	0,20	0,35
234	0,30	0,15
278	0,30	0,20
295	0,30	0,25
336	0,30	0,30
206	0,30	0,35
267	0,40	0,15
196	0,40	0,20
199	0,40	0,25
277	0,40	0,30
213	0,40	0,35
208	0,50	0,15
245	0,50	0,20

273	0,50	0,25
189	0,50	0,30
286	0,50	0,35

Table 8.10. Variation of parameters in random/AntNet (200 containers)

best tour length (in ms)	α	C_2
368	0,20	0,15
257	0,20	0,20
178	0,20	0,25
215	0,20	0,30
201	0,20	0,35
290	0,30	0,15
262	0,30	0,20
405	0,30	0,25
395	0,30	0,30
234	0,30	0,35
342	0,40	0,15
205	0,40	0,20
207	0,40	0,25
255	0,40	0,30
230	0,40	0,35
214	0,50	0,15
268	0,50	0,20
387	0,50	0,25
274	0,50	0,30
379	0,50	0,35

Table 8.11. Combination in star topology.

combination	time in ms
AntNet/AntNet	360000
AntNet/BeeAlg.	363000
AntNet/GA	361000
AntNet/MMAS	358000
AntNet/RoundRobin	361000
AntNet/Sender	360000
BeeAlg./AntNet	409000
BeeAlg./BeeAlg.	346000
BeeAlg./GA	413000
BeeAlg./MMAS	404000
BeeAlg./RoundRobin	410000
BeeAlg./Sender	410000
GA/AntNet	346000
GA/BeeAlg.	366000
GA/GA	357000
GA/MMAS	354000
GA/RoundRobin	356000
GA/Sender	355000
MMAS/AntNet	354000
MMAS/BeeAlg.	359000
MMAS/GA	359000
MMAS/MMAS	349000
MMAS/RoundRobin	351000
MMAS/Sender	352000
RoundRobin/AntNet	400000
RoundRobin/BeeAlg.	404000
RoundRobin/GA	408000
RoundRobint/MMAS	397000
RoundRobin/RoundRobin	401000
RoundRobin/Sender	412000
Sender/AntNet	881000
Sender/BeeAlg.	880000
Sender/GA	859000
Sender/MMAS	867000
Sender/RoundRobin	878000
Sender/Sender	847000

Table 8.12. Combinations in chain topology.

combination	time in ms
AntNet/AntNet	107000
AntNet/BeeAlg.	107000
AntNet/GA	101000
AntNet/MMAS	103000
AntNet/RoundRobin	112000
AntNet/Sender	106000
BeeAlg./AntNet	96000
BeeAlg./BeeAlg.	113000
BeeAlg./GA	104000
BeeAlg./MMAS	104000
BeeAlg./RoundRobin	94000
BeeAlg./Sender	88000
GA/AntNet	94000
GA/BeeAlg.	93000
GA/GA	94000
GA/MMAS	96000
GA/RoundRobin	96000
GA/Sender	131000
MMAS/AntNet	120000
MMAS/BeeAlg.	115000
MMAS/GA	101000
MMAS/MMAS	88000
MMAS/RoundRobin	332000
MMAS/Sender	113000
RoundRobin/AntNet	372000
RoundRobin/BeeAlg.	381000
RoundRobin/GA	384000
RoundRobint/MMAS	408000
RoundRobin/RoundRobin	371000
RoundRobin/Sender	372000
Sender/AntNet	374000
Sender/BeeAlg.	383000
Sender/GA	379000
Sender/MMAS	404000
Sender/RoundRobin	371000
Sender/Sender	308000

Table 8.13. Combination in full topology.

combination	time
AntNet/AntNet	345000
AntNet/BeeAlg.	348000
AntNet/GA	346000
AntNet/MMAS	346000
AntNet/RoundRobin	343000
AntNet/Sender	343000
BeeAlg./AntNet	370000
BeeAlg./BeeAlg.	376000
BeeAlg./GA	393000
BeeAlg./MMAS	386000
BeeAlg./RoundRobin	382000
BeeAlg./Sender	398000
GA/AntNet	338000
GA/BeeAlg.	338000
GA/GA	340000
GA/MMAS	341000
GA/RoundRobin	338000
GA/Sender	360000
MMAS/AntNet	360000
MMAS/BeeAlg.	350000
MMAS/GA	359000
MMAS/MMAS	352000
MMAS/RoundRobin	358000
MMAS/Sender	359000
RoundRobin/AntNet	77000
RoundRobin/BeeAlg.	76000
RoundRobin/GA	81000
RoundRobint/MMAS	82000
RoundRobin/RoundRobin	116000
RoundRobin/Sender	90000
Sender/AntNet	300000
Sender/BeeAlg.	306000
Sender/GA	301000
Sender/MMAS	296000
Sender/RoundRobin	320000
Sender/Sender	303000

Table 8.14 Combination in ring topology.

combination	time
AntNet/AntNet	104000
AntNet/BeeAlg.	104000
AntNet/GA	101000
AntNet/MMAS	99000
AntNet/RoundRobin	102000
AntNet/Sender	103000
BeeAlg./AntNet	102000
BeeAlg./BeeAlg.	103000
BeeAlg./GA	104000
BeeAlg./MMAS	105000
BeeAlg./RoundRobin	107000
BeeAlg./Sender	93000
GA/AntNet	103000
GA/BeeAlg.	100000
GA/GA	105000
GA/MMAS	100000
GA/RoundRobin	105000
GA/Sender	103000
MMAS/AntNet	113000
MMAS/BeeAlg.	108000
MMAS/GA	110000
MMAS/MMAS	142000
MMAS/RoundRobin	93000
MMAS/Sender	105000
RoundRobin/AntNet	398000
RoundRobin/BeeAlg.	409000
RoundRobin/GA	406000
RoundRobint/MMAS	398000
RoundRobin/RoundRobin	412000
RoundRobin/Sender	405000
Sender/AntNet	305000
Sender/BeeAlg.	316000
Sender/GA	296000
Sender/MMAS	298000
Sender/RoundRobin	302000
Sender/Sender	299000

REFERENCES

[AACH00] Abelson H., Allen D., Coore D., Hanson C., Homsy G., Knight T., Nagpal R., Rauch E., Sussman G. and Weiss R., *Amorphous Computing*, Communication of the ACM, vol. 43 (5), pp. 74–82, 2000.

[Aber01] Aberer K., *P-grid: A self-organizing access structure for P2P information systems*, 9th International Conference on Cooperative Information Systems, Springer-Verlag, pp. 179–194, 2001.

[ACDDHSP03] Aberer K., Cudré-Mauroux P., Datta A., Despotovic Z., Hauswirth M., Punceva M. and Schmidt, R., *P-Grid: a self-organizing structured P2P system,* SIGMOD Rec., vol. 32(3), pp. 29-33, 2003.

[AGKT02] Andrzejak A., Graupner S., Kotov V. and Trinks H., *Algorithms for Self-Organization and Adaptive Service Placement in Dynamic Distributed Systems*, Tech. Rep. HPL-2002-259, Hewlett-Packard Labs Palo Alto, 2002.

[AlBa02] Albert R. Barabási A.-L., *Statistical mechanics of complex networks*, Reviews of Modern Physics, vol. 74, pp. 47–97, 2002.

[Alha04] Alhazbi S.M., *Measuring the complexity of Component-Based System Architecture*, Int. Conf. on Information and Communication Technologies: From Theory to Applications, 2004.

[AnSp04] Androutsellis-Theotokis S., and Spinellis D., *A survey of peer-to-peer content distribution technologies*, ACM Comput. Survey, vol. 36, pp. 335-371, 2004.

[ApBu05] Apel S. and Buchmann E., *Biology-Inspired Optimizations of Peer-to-Peer Overlay Networks*, Quellenangabe Praxis der Informationsverarbeitung und Kommunikation, vol. 28(4), 2005.

[Ashb47] Ashby W.R., *Principles of the Self-Organizing Dynamic System*, Journal of General Psychology, vol.37, pp. 125-128, 1947.

[BaBD02] Backström T., Bjerlöv M. and Docherty P., *Organization of work in complex environments – a continuous development through communication and learning*, Managing the Complex IV, 2002.

[BaBK02] Balazinska M., Balakrishnan H. and Karger D., *Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery*, LNCS, Springer, pp. 149–153, 2002.

[BaCh03] Barker. K. J. and Chrisochoides N. P., *An evaluation of a framework for the dynamic load balancing of highly adaptive and irregular parallel applications*, ACM/IEEE conference on Supercomputing, 2003.

[BaJe08] Babaoglu O. and Jelasity M., *Self-* properties through gossiping*. Philosophical Transactions of the Royal Society; vol. 366(188), pp. 13747–3757, 2008.

[Ball04] Ball P., Critical Mass: How One Thing Leads to Another, ISBN 0-434-01135-5, 2004.

[Bane92] Banerjee A. V., *A Simple Model of Herd Behavior*, The Quarterly Journal of Economics, vol. 107(3), pp. 797-817, 1992.

[BBJM07] Babaoglu O., Binci T., Jelasity M. and Montresor A., *Firefly-inspired heartbeat synchronization in overlay networks*, 1st IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007), 2007.

[BCDDD05] Babaoglu O., Canright G., Deutsch A., Di Caro G., Ducatelle F., Gambardella L., Ganguly N., Jelasity M., Montemanni R. and Montesor A., *Design patterns from biology for distributed computing*, European Conference on Complex Systems, 2005.

[BCCP04] Barker K., Chernikov A., Chrisochoides N. and Pingali K., *A load balancing framework for adaptive and asynchronous applications*, IEEE Transactions on Parallel and Distributed Systems, vol. 15(2), pp.183-192, 2004.

[BeBM93a] Beasley D., Bull D.R. and Martin R.R., *An overview of genetic algorithms: fundamentals*, University Computing, vol. 15(2), pp. 58-69, 1993.

[BeBM93b] Beasley D., Bull D.R. and Martin R.R., *An overview of genetic algorithms: research topics*, University Computing, vol. 15(4), pp. 170-181, 1993.

[Berk74] Berk R., A. Collective Behavior, Dubuque, Iowa: Wm. C. Brown, 1974.

[Bezd81] Bezdek J. C., *Pattern Recognition with Fuzzy Objective Function Algorithms*, Plenum Press, New York, 1981.

[BFKMT10] Bessler S., Fischer A., Kühn E., Mordinyi R. and Tomic S., *Using Tuple-Spaces to manage the Storage and Dissemination of Spatial-temporal Content*, Journal of Computer and System Sciences, Elsevier, 2010.

[BHLTT09] Bray T., Hollander D., Layman A., Tobin R. and Thompson H.S., *Namespaces in XML 1.0*, W3C Recommendation, December 2009.

[BiHW92] Bikhchandani S., Hirshleifer D. and Welch I., *A Theory of Fads, Fashion, Custom, and Cultural Change as Informational Cascades*, Journal of Political Economy, vol. 100(5), pp. 992-1026, 1992.

[Blem03] Blem C., *Beam-ACO Hybridizing ant colony optimization with beam search-an application to open shop scheduling,* Technical report TR/IRIDIA/2003-17, 2003.

[BlGu03] Blass A. and Gurevich Y., *Algorithms: A Quest for Absolute Definitions*, Bulletin of European Association for Theoretical Computer Science 81, 2003.

[BLLNL09] Bastos Filho C., de Lima Neto F., Lins A. , Nascimento A. and Lima M. , *Fish School Search*, Nature-Inspired Algorithms for Optimisation, Studies in Computational Intelligence, vol. 193, 2009.

[BlRo03] Blum, C. and Roli A*., Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison*, ACM Computing Surveys, vol.35(3), pp. 268–308, 2003.

[BoCH00] Bojinov H., Casal A. and Hogg T., *Emergent Structures in Modular Self-Reconfigurable Robots*, Proceedings of the International Conference on Robotics and Automation, IEEE CS Press, 2000.

[BoDT99] Bonabeau E., Dorigo M. and Theraulaz G., *Swarm Intelligence: From Natural to Artificial Systems*, Santa Fe Inst. Studies in the Sc. of Complexity Series, Oxford Press, 1999.

[Bond00] Bondi A.B, *Characteristics of scalability and their impact on performance*, 2nd Int. Workshop on Software and Performance, ACM, 2000.

[BrMe008] Bronevich A.G. and Meyer W., *Load balancing algorithms based on gradient methods and their analysis through algebraic graph theory*, Journal Parallel Distributed Computing, vol. 68, pp. 209-220, 2008.

[BrRP08] Brinkschulte, U., von Renteln, A., and Pacher, M., *Measuring the quality of an artificial hormone system based task mapping*, 2nd Int. Conf. on Autonomic Computing and Communication Systems, 2008.

[BrTy00] Bradley D. and Tyrrell A., *Immunotronics: Hardware fault tolerance inspired by the immune system,* 3rd International Conference on Evoluable Systems (ICES2000), 2000.

[Brunn01] Brunnermeier M. K., *Asset Pricing under Asymmetric Information: Bubbles, Crashes, Technical Analysis, and Herding*, Oxford University Press 2001.

[BTDAC97] Bonabeau E., Theraulaz G., Deneubourg J.L., Aron S. and Camazine S., *Self-organization in social insects*, Trends in Ecology and Evolution, vol. 12, pp. 188-193, 1997.

[BTFGGDM02] Ben-Yehuda S., Tolasch T., Francke W., Gries R., Gries G., Dunkelblum D. and Mendel Z*., Aggregation pheromone of the almond bark beetle Scolytus amygdali (Coleoptera: Scolytidae)*, IOBC wprs Bulletin, vol. 25, 2002.

[BuDB09] Budayan C., Dikmen I. and Birgonul M. T., *Comparing the performance of traditional cluster analysis, self-organizing maps and fuzzy C-means method for strategic grouping*, Expert System Applications, vol. 36 (9), 2009.

[BuHu97] Buchanan D., Huczynski A., Organisational behaviour, Prentice Hall, 3rd Edition, 1997.

[CaDa03] Cao Y. and Dasgupta D., *An Immunogenetic Approach in Chemical Spectrum Recognition*, Advances in Evolutionary Computing (Ghosh & Tsutsui, eds.), 2003.

[CaDFSTB03] Camazine S., Deneubourg J., Franks N.R., Sneyd J., Theraulaz G. and Bonabeau E., *Self-Organization in Biological Systems,* Princeton University Press, 2003.

[CaDo98] Caro G. D. and Dorigo M., *Extending AntNet for best-effort quality-of-service routing*, 1st Int. Workshop on Ant Colony Optimization (ANTS'98), 1998.

[CaGe89] Carriero N. and Gelernter D., *Linda in Context*, CACM, vol. 32 (4), pp. 444-458, 1989.

[Cald07] Caldarelli G., *Scale-Free Networks*, Oxford University Press, Oxford, 2007.

[CaSn91] Camazine S. and Sneyd J., *A model of collective nectar source selection by honey bees: Self-organization through simple rules*, Journal of Theoretical Biology, vol. 149(4), pp. 547-571, 1991.

[CaVi08] Casadei M and Viroli M., *Applying Self-Organizing Coordination to Emergent Tuple Organization in Distributed Networks*, 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO'08), 2008.

[CDFSTB01] Camazine S., Deneubourg J., Franks N.R., Sneyd J., Theraulaz G. and Bonabeau E., *Self-Organization in Biological Systems*, Princeton University Press, 2001.

[ChLH08] Chen J.C., Liao G.X., Hsie J.S. and Liao C., *A study of the contribution made by evolutionary learning on dynamic load-balancing problems in distributed computing systems*, Expert Syst. Appl., vol. 34, pp. 357-365, 2008.

[CHSW02] Clarke I., Hong T., Sanberg, O., and Wiley, B., *Protecting free expression online with Freenet,* IEEE Internet Computing, vol. 6(1), pp. 40–49, 2002.

[ChZC10] Chen W. N., Zhang J. and Chung H., *Optimizing Discounted Cash Flows in Project Scheduling — An Ant Colony Optimization Approach*, IEEE Trans. on Systems, Man, and Cybernetics, 40(5), pp.64-77, 2010.

[Clar93] Clark F. O., *Parallel Algorithms for Hierarchical Clustering,* Parallel Computing, vol. 21, pp. 1313-1325, 1993.

[CLRS09] Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C., *Introduction to Algorithms*, The MIT Press, 2009.

[ClSW00] Clarke, I., Sandberg, O., and Wiley, B., *Freenet: A distributed anonymous information storage and retrieval system*, Workshop on Design Issues in Anonymity and Unobservability, 2000.

[CMVT07a] Casadei M., Menezes R., Viroli M. and Tolksdorf R., *A Self-organizing Approach to Tuple Distribution in Large-Scale Tuple-Space Systems*, International Workshop on Self-Organizing Systems, IWSOS'07, 2007.

[CMVT07b] Casadei M., Menezes R., Viroli M. and Tolksdorf R., *Self-organized over-clustering avoidance in tuple-space systems*, IEEE Congress on Evolutionary Computation, 2007.

[CoGD97] Commons M.L., Goodheart E.A. and Dawson T.L., *Psychophysics of Stage: Task Complexity and Statistical Models*, International Objective Measurement Workshop at the Annual Conference of the American Educational Research Association, 1997.

[Cohe10] Cohen U., *Inside gigaspaces XAP - technical overview and value proposition*, Technical report, GigaSpaces White Paper, Last accessed: August, 2010.

[CRCSL02] Cortes A., Ripolli A., Cedo F., Senar, M. A. and Luque, E., *An asynchronous and iterative load balancing algorithm for discrete load model*, Journal Parallel Distributed Computing, vol. 62, pp. 1729-1746, 2002.

[CrKS09] Craß S., Kühn E., and Salzer G., *Algebraic Foundation of a Data Model for an Extensible Space-Based Collaboration Protocol*, 13th Int. Database Engineering & Applications Symposium (IDEAS), 2009.

[CSLG06] Chong C. S., Sivakumar A. I., Low, M. Y., and Gay K. L., *A bee colony optimization algorithm to job shop scheduling*, 38th Conf. on Winter Simulation, 2006.

[CuWa09] Cui L. and Wang H., *Reachback Firefly Synchronicity with Late Sensitivity Window in Wireless Sensor Networks*, 9th International Conference on Hybrid Intelligent Systems, 2009.

[DaFo96] Dasgupta D. and Forrest S., *Novelty Detection in Time Series Data using Ideas from Immunology*, ISCA 5th International Conference on Intelligent Systems, 1996.

[DAGP90] Deneuborg J.-L., Aron S., Goss S. and Pasteels J.-M., *The self-organizing exploratory pattern of the Argentine ant,* Jourval of Insect Behaviour, vol.3, pp.159-168, 1990.

[DaJS06] Davidsson P., Johansson S. and Svahnberg M., *Characterization and evaluation of multi-agent systems architectural styles,* Software Engineering for Multi-Agent Systems IV, vol. 3914, pp. 179-188, 2006.

[Dasg99] Dasgupta D. (Ed.), *Artificial Immune Systems and Their Applications*, Springer-Verlag, 1999.

[Davi85] Davis L., *Job Shop Scheduling with Genetic Algorithms*, 1st International Conference on Genetic Algorithms, 1985.

[Dawi96] Dawid, H., *Adaptive Learning by Genetic Algorithms: Analytical Results and Applications to Economic Models*, Springer-Verlag New York, Inc. 1996.

[DDFG06] Dobson S., Denazis S., Fernández A., Gaïti D., Gelenbe E., Massacci F., Nixon P., Sare F., Schmidt N. and Zambonelli F., *A survey of autonomic communications*, ACM Trans. Auton. Adapt. Syst., vol.1(2), pp: 223-259, 2006.

[DeLR77] Dempster A.P., Laird N.M., and Rubin D.B., *Maximum Likelihood from Incomplete Data via theEM algorithm*, Journal of the Royal Statistical Society, vol. 39(1), pp.1-38, 1977.

[DeSD00] Den Bseten M., Stützle T. and Dorigo M., *Ant colony optimization for the total weighted tardiness problem,* 6th International Conference on Parallel Problem Solving from Nature, pp.611-620, 2000.

[DiDo98a] Di Caro G. and Dorigo M., *AntNet: Distributed Stigmergetic Control for Communications Networks*, Journal of Artificial Intelligence Research (JAIR), vol. 9, pp. 317-365, 1998.

[DiDo98b] Di Caro G. and Dorigo M., *Extending AntNet for best-effort quality-of-service routing*, 1st International Workshop on Ant Colony Optimization (ANTS), 1998.

[DiFM00] Dingledine R., Freedman M., and Molnar D., *Peer-to-peer: Harnessing the power of disruptive technology*, 1st Ed. O'Reilly (Chapter 16), Accountability, pp. 271–340, 2000.

[DIGK05] Di Marzo Serugendo G., Gleizes M. and Karageorgos A., *Self-organization in multi-agent systems*, The Knowledge Engineering Review, vol. 20, pp. 165-189, 2005.

[DoSt05] Dorigo M. and Stützle T., *Ant Colony Optimization*, MIT Press, 2005.

[DrRo01] Druschel P. and Rowstron A., *Past: A largescale, persistent peer-to-peer storage utility*, 8th Workshop on Hot Topics in Operating Systems, 2001.

[DSCB03] Da Silva D.P., Cirne W., Brasileiro F.V. and Grande C., *Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks*, Applications on Computational Grids, Euro-Par 2003 Parallel Processing, LNCS, pp. 169-180, 2003.

[FaRB98] Fayyad, U., Reina, C. and Bradley, P. S., *Initialization of iterative refinement clustering algorithms,* 4th International Conference on Knowledge Discovery and Data Mining, 1998.

[Farz09] Farzi S., *Efficient Job Scheduling in Grid Computing with Modified Artificial Fish Swarm Algorithm*, International Journal of Computer Theory and Engineering, vol. 1 (1), 2009.

[Floy67] Floyd R.W., *Nondeterministic Algorithms*, Journal of the ACM, vol. 14(4), pp. 636-644, 1967.

[Floy05] Floyd S., *Adaptive Web Cache*, http://www.icir.org/floyd/web.html, 2005.

[FoCM02] Focardi S., Cincotti S., Marchesi M., Self-organization and market crashes, Journal of Economic Behavior and Organization, vol. 49(2), pp.241– 267, 2002.

[FoHo03] Fortnow L. and Homer S., *A Short History of Computational Complexity*, Bulletin of the EATCS, vol.80, pp. 95–133, 2003.

[FrHA99] Freeman E., Hupfer S. and Arnold K., *JavaSpaces: Principles, Patterns and Practices*, Addison-Wesley, 1999.

[Fuch03] Fuchs C., *Co-operation in Complex, Self-Organising, Information-Generating Systems*, 47th Annual Conference of the International Society for the Systems Sciences (ISSS), 2003.

[GeCa92] Gelernter D. and Carriero N., *Coordination languages and their significance*, ACM Communication, vol. 35, pp. 97-107, 1992.

[Gele85] Gelernter D., *Generative communication in Linda*, ACM Transactions on Programming Languages and Systems, vol.7(1), pp. 80–112, 1985.

[GeRa03] Georgousopoulos C. and Rana O. F., *Combining state and model-based approaches for mobile agent load balancing,* ACM/ SAC '03 symposium on Applied Computing, 2003.

[Gian06] Giannantoni C., *Mathematics for generative processes: Living and non-living systems*, Journal of Computational Applied Mathematics, vol. 189(1-2), pp. 324-340, 2006.

[GLSKS04] Godfrey B., Lakshminarayanan K., Surana S., Karp R. and Stoica I., *Load balancing in dynamic structured P2P systems*, IEEE INFOCOM, 2004.

[Gold89] Goldberg D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley Publ, 1989.

[GoMO06] Gomez-Ballester E., Mico L., and Oncina J., Some *approaches to improve tree-based nearest neighbour search algorithms*, Pattern Recognition, vol. 39(2), pp.171-179, 2006.

[GoSc01] Gomoluch J. and Schroeder M., *Information agents on the move: A survey with load balancing with mobile agents*, Software Focus, vol. 2(2), 2001.

[Good01] Goodrich M., *Algorithm Design: Foundations, Analysis, and Internet Examples*, Wiley, 2001.

[Grass86] Grassberger P., *Toward a quantitative theory of self-generated complexity*, International Journal of Theoretical Physics, vol. 25, pp. 907-938, 1986.

[GrMT08] Graff D., Menezes R. and Tolksdorf R., *On the performance of swarm-based tuple organization in LINDA systems*, IEEE Congress on Evolutionary Computation, 2008.

[HaGr02] Hardaker G. and Graham G., *Energizing your e-commerce through self-organising collaborative marketing networks*, Technical report, School of Business, University of Salford, UK, 2002.

[HCCD05] Hu T., Chen G., Chen K. and Dong J., *An adaptive load balancing framework for parallel database systems based on collaborative agents*, vol. 1, pages 464-468, 9th International Conference on Computer Supported Cooperative Work in Design, 2005.

[HDKC06] Hassas S., Di Marzo-Serugendo G., Karageorgos A. and Castelfranchi C., *On Self-Organising Mechanisms from Social*, Business and Economic Domains, Informatica, vol. 30, pp. 63-71, 2006.

[HeBP07] Herrero P., Bosque J.L. and Perez M.S., *An agents-based cooperative awareness model to cover load balancing delivery in grid environments*, OTM Workshops (1), 2007.

[HeGe03] Heylighen F. and Gershenson C., *The Meaning of Self-organization in Computing*, IEEE Intelligent Systems, section Trends & Controversies - Self-organization and Information Systems, 2003.

[Herr03] Herrmann K., *MESH Mdl — A Middleware for Self-Organization in Ad Hoc Networks,* 23rd International Conference on Distributed Computing Systems, 2003.

[Heyl01] Heylighen F., *The Science of Self-Organization and Adaptivity*, in: L. D. Kiel, (ed.) Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems, EOLSS Publishers, Oxford, 2001.

[Heyl08] Heylighen F., *Complexity and Self-Organization*, in: M. J. Bates and M. N. Maack (eds.), The Encyclopedia of Library and Information Science, Taylor & Francis, 2008.

[HoEw07] Ho C. and Ewe H., *Ant colony optimization approaches for the dynamic load-balanced clustering problem in ad hoc networks*, Swarm Intelligence Symposium, SIS'2007 IEEE, 2007.

[HoFo00] Hofmeyr S.A. and Forrest S., *Architecture for an artificial immune system*, Evolutionary Computation, vol. 8(4), pp. 443–473, 2000.

[HoWo03] Hohpe G. and Woolf B., *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*, Addison-Wesley, 2003.

[HRTB00] Hadji R., Rahoual M., Talbi E. and Bachelet V., *Ant colonies for the set covering problem,* ANTS, 2000.

[InGE00] Intanagonwiwat C., Govindan R. and Estrin D., *Directed diffusion: A scalable and robust communication paradigm for sensor networks*, Proceedings of the International Conference on Mobile Computing and Networking, ACM Press, 2000.

[JaTh07] Jansen T. and Theile M., *Stability in the self-organized evolution of networks*, 9th Annual Conference on Genetic and Evolutionary Computation, 2007.

[JoDK02] Johansson S., Davidsson P. and Kristell M., *Four multi-agent architectures for intelligent network load management*, 4th International Workshop on Mobile Agents for Telecommunication Applications (MATA), 2002.

[JoLi10] Joung Y. and Lin Z, *On the self-organization of a hybrid peer-to-peer system,* Journal of Network and Computer Applications, vol. 33(2), pp. 183-202, 2010.

[JoMa03] Jones C. and Mataric M., *From Local to Global Behavior in Intelligent Self-Assembly*, Proceedings of the Conference on Robotics and Automation, IEEE Press, 2003.

[JoWo00] Jogalekar P. and Woodside C.M., *Evaluating the Scalability of Distributed Systems*, IEEE Trans. on Parallel Distributed Systems, vol. 11, pp. 589-603, 2000.

[KaBo04] Kaner C. and Bond W.P, *Software Engineering Metrics: What Do They Measure and How Do We Know?*, 10th International Software Metrics Symposium, METRICS, 2004.

[Kant1892] Kant I., *Critique of Judgment*, Translated by J. H. Barnard, New York: Hafner Publishing, 1951. (Original publication date 1892)

[KaRu04] Karger D.R. and Ruhl M., *Simple efficient load balancing algorithms for peer-to-peer systems*, 16th annual ACM symposium on Parallelism in Algorithms and Architectures, SPAA '04, 2004.

[KFŠT96] Kratica J., Filipović V., Šešum V. and Tošić D., *Solving of the Uncapacitated Warehouse Location Problem Using a Simple Genetic Algorithm,* 14th International Conference on Material Handling and Warehousing, pp. 3.33-3.37, 1996.

[KLŠF98] Kratica J., Ljubić I., Šešum V. and Filipović V., *Some methods of solving the travelling salesperson problem using genetic algorithms,* 2nd International Symposiumof Industrial Engineering SIE'98, pp. 281-284, 1998.

[KMGBT09] Kühn E., Mordinyi R, Goiss H.D, Moser T., Bessler S. and Tomic S., *Using tuple-spaces to build a storage P2P system for structured and dynamic data,* 2nd Int. Workshop on Adaptive Systems in Heterogeneous Environments, IEEE, Japan, 2009.

[KMKS09] Kühn E., Mordinyi R., Keszthelyi L. and Schreiber C., *Introducing the Concept of Customizable Structured Spaces for Agent Coordination in the Production Automation Domain*, 8th International Conference on Autonomous Agents and Multiagent Systems, 2009.

[KrNe99] Krishnakumar K. and Neidhoefer J., *Immunized Adaptive Critic for an Autonomous Aircraft Control Application*, Artificial Immune Systems and Their Applications, 1999.

[KrRŠ97] Kratica J., Radojević S. and Šešum V., *A method of improving the execution time of simple genetic algorithm,* 23rd Jupiter Conference, pp. 457-462, 1997.

[Krug96] Krugman P., *The Self-Organizing Economy*, Blackwell Publishers, 1996.

[KTTRS09] Ke L., Thomas K., Torres C. E., Rossi L.F. and Shen C., *Naturally Adaptive Protocol for Wireless Sensor Networks Based on Slime Mold*, 3rd IEEE International Conference on Self-Adaptive and Self-Organizing Systems, 2009.

[Kühn94] Kühn E., *Fault-tolerance for communicating multidatabase transactions,* 27th Hawaii International Conference on System Sciences (HICSS), 1994.

[Kühn01] Kühn E., *Virtual Shared Memory for Distributed Architectures*, Nova Science Publ., 2001.

[KuKG08] Kumar A., Kumar R. and Grover P. S., *Towards a Unified Framework for Complexity Measurement in Aspect-Oriented Systems*, International Conference on Computer Science and Software Engineering, 2008.

[KüMS08] Kühn E., Mordinyi R. and Schreiber C., *An Extensible Space-based Coordination Approach for Modeling Complex Patterns in Large Systems*, 3rd International Symposium on Leveraging Appl. on Formal Methods, Verification and Validation, 2008.

[KüRJ05] Kühn E., Riemer J. and Joskowicz G., *XVSM (eXtensible Virtual Shared Memory) Architecture and Application*, Technical Report TU-Vienna, E185/1, 2005.

[KüRŠ07] Kühn E., Ruhdorfer A. and Šešum-Cavic V., *Asynchronous replication conflict classification, detection and resolution for heterogeneous data*, International Conference of Software and Data Technology, (ICSOFT), 2007.

[KüŠe09] Kühn E. and Šešum-Cavic V., *A Space-Based Generic Pattern for Self-Initiative Load Balancing Agents*, Engineering Societies in the Agents World ESAW, LNAI Springer Verlag, 2009.

[Lamp09] Lamport L., *The PlusCal Algorithm Language*, Theoretical Aspects of Computing-ICTAC, LNCS 5684, pp. 36-60, 2009.

[LDTN08] Lemmens N., De Jong S., Tuyls K. and Nowé A., *Bee Behaviour in Multi-agent Systems*, Adaptive Agents and MAS III, LNCS, pp. 145-156, 2008.

[Leit95] Leitch R.D., *Reliability Analysis for Engineers*. Oxford University Press, 1995.

[LiDW09] Liu B., Ding Y. and Wang J., *A collaborative optimized genetic algorithm based on regulation mechanism of neuroendocrine-immune system*, Genetic and Evolutionary Computation, GEC '09, 2009.

[LiKe87] Lin F.C. and Keller R.M., *The gradient model load balancing method*, IEEE Trans. On Software Engineering, vol. 13, pp. 32-38, 1987.

[LiSm04] Liang Y. C. and Smith A. E., *An ant colony optimization algorithm for the redundancy allocation problem (RAP),* IEEE Trans. on Reliability, vol. 53(3), pp. 417-423, 2004.

[Lloy01] Lloyd S., *Measures of Complexity: A Nonexhaustive List,* IEEE Control System, 2001.

[LMLPW00] Lambrinos D., Moeller R., Labhart T., Pfeifer R. and Wehner R., *A mobile robot employing insect strategies for navigation*, Robotics and Autonomous Systems, vol. 30 (1-2), pp. 39-64, 2000.

[LSTD01] Lioni, Sauwens C., Theraulaz G. and Deneubourg J.L., *Chain formation in Oecophylla longinoda*, Journal of Insect Behavior, vol. 14, pp. 679–696, 2001.

[LTSS02] Ledlie J., Taylor J. M., Serban L. and Seltzer M. I*., Self-organization in peer-to-peer systems*, ACM SIGOPS European Workshop, 2002.

[MaBi02] Maymounkov P. and Birman K., *Kademlia: A Peer-to-peer Information System Based on the XOR Metric*, 1st International Workshop on Peer-to-Peer Systems (IPTPS '02), 2002.

[Macl04] MacLennan, B. J., *Applications of Self-Organization to Command, Control, and Coordination: A Position Paper*; UT CS Dept. TR UT-CS-04-534, 2004.

[Masa09] Masatoshi S., *dRuby and Rinda: Implementation and application of distributed Ruby and its parallel coordination mechanism*, International Journal of Parallel Programming, vol. 37(1), pp. 37–57, 2009.

[MaTA07] Markovic G., Teodorovic D. and Acimovic-Raspopovic V., *Routing and wavelength assignment in all-optical networks based on the bee colony optimization*, AI Communications, vol. 20 (4), pp. 273-285, 2007.

[MaZa04] Mamei M. and Zambonelli F., *Programming Pervasive and Mobile Computing Applications with the TOTA Middleware*, Proceedings of the International Conference On Pervasive Computing (Percom), IEEE CS Press, 2004.

[MaZL04] Mamei M., Zambonelli F. and Leonardi L., *Co-Fields: A Physically Inspired Approach to Distributed Motion Coordination*, IEEE Pervasive Computing, vol. 3(2), pp. 52–61, 2004.

[MeKo05] Meer H. de, and Koppen C., *Self-Organization in Peer-to-Peer Systems*, Peer-to-Peer Systems and Applications, pp. 247-266, 2005.

[Mene05] Menezes R., *Self-Organization and Computer Security: A Case Study in Adaptive Coordination*, ACM Symposium on Applied Computing, 2005.

[MeTo03] Menezes R. and Tolksdorf R., *A New Approach to Scalable Linda-systems Based on Swarms*, Proceedings of ACM SAC, pp. 375–379, 2003.

[MiSt90] Mirollo R. and Strogatz S., *Synchronization of pulse-coupled biological oscillators*, SIAM Journal on Applied Mathematics, vol. 50 (6), pp. 1645-1662, 1990.

[MMBR09] Mullen R.J., Monekosso D., Barman S. and Remagnino P., *A review of ant algorithms*, Expert Systems with Applications, vol. 36, pp. 9608–9617, 2009.

[MMTZ06] Mamei M., Menezes R., Tolksdorf R. and Zambonelli F., *Case studies for self-organization in computer science,* Journal of System Architecture, vol. 52(8-9), pp.443-460, 2006.

[MoKS10] Mordinyi R., Kühn E. and Schatten A., *An Architectural Framework for Agile Software Development*, 11th International Conference on Agile Software Development (XP), 2010.

[MoMa08] Monismith, D.R. and Mayfield, B.E., *Slime Mold as a model for numerical optimization*, Swarm Intelligence Symposium SIS2008, 2008.

[Mord10] Mordiny R., *Managing Complex and Dynamic Software Systems with Space-Based Computing*, PhD Thesis, 2010.

[MTIK06] Murata Y., Takizawa H., Inaba T. and Kobayashi H., *A distributed and cooperative load balancing mechanism for large-scale P2P systems*, SAINT-W '06:International Symposium on Applications on Internet Workshops, 2006.

[NaTo04] Nakrani S. and Tovey C., *On Honey Bees and Dynamic Server Allocation in Internet Hosting Centers*, Adaptive Behaviour, vol. 12(3-4), pp. 223-240, 2004.

[NeSR06] Nezamabadi-pour H., Saryazdi S., and Rashedi E., *Edge detection using ant algorithms, Soft Computing*, vol. 10(7), pp. 623-628, 2006.

[NWQDS03] Nejdl W., Wolf B., Qu C., Decker S., Sintek M., Naeve A., Nilsson M., Palmer M., and Risch T., *Edutella: A P2P networking infrastructure based on RDF*, 12th International Conference on World Wide Web, 2003.

[OlPu06] Olague G. and Puente C., *The Honeybee Search Algorithm for Three-Dimensional Reconstruction*, 8th European Workshop on Evolutionary Computation in Image Analysis and Signal Processing, LNCS 3907, 2006.

[OsLa96] Osman, I.H. and Laporte G., *Metaheuristics:A bibliography*, Ann. Operational Research, vol. 63, pp. 513–623, 1996.

[PaAr98] Papadopoulos G. A. and Arbab F., *Coordination Models and Languages*, Advances in Computers, Academic Press, pp. 329-400, 1998.

[PaBS04] Parunak V., Brueckner S. and Sauter J., *Digital Pheromones for Coordination of Unmanned Vehicles*, Proceedings of the Workshop on Environments for Multi-agent Systems, LNAI 3374, Springer Verlag, 2004.

[PaLF02] Parpinelli R. S., Lopes H. S. and Freitas A., *Data mining with an ant colony optimization algorithm,* IEEE Transaction on Evolutionary Computation, vol. 6(4), pp. 321-332, 2002.

[PhSG2006] Pham D.T., Soroka A.J., Ghanbarzadeh A., Koç E., Otri S. and Packianather M., *Optimising neural networks for identification of wood defects using the Bees Algorithm*, International Conference on Industrial Informatics, IEEE, 2006.

[PiMR99] Picco G.P., Murphy A.L. and Roman G., *Lime: Linda meets mobility*, ICSE '99, 21st International Conference on Software engineering, 1999.

[PKLP07] Pham D.T., Koç E., Lee J.Y. and Phrueksanant J., *Using the Bees Algorithm to schedule jobs for a machine*, 8th International Conference on Laser Metrology, 2007.

[POAKS09] Pussep K., Oechsner S., Abboud O., Kantor M. and Stiller B., *Impact of Self-Organization in P2P Overlays on Underlay Utilization*, 4th International Conference on Internet and Web Applications and Services, ICIW '09, 2009.

[Pola03] Polani D., *Measuring Self-Organization via Observers*, ECAL'03, 2003.

[PoMe08] Roach C. and Menezes R., *Handling Dynamic Networks Using Evolution in Ant-Colony Optimization*, 21st International Conference on Industrial, Engineering and Other Appl. of Applied Intelligent Systems, 2008.

[Poor01] Poor R., *Embedded Networks: Pervasive, Low-Power, Wireless Connectivity*, PhD Thesis, MIT, 2001.

[Potv96] Potvin J., *Genetic algorithms for the traveling salesman problem*, Annals of Operational Research, vol. 63(3), 1996.

[Putr03] Putrycz E., *Design and implementation of a portable and adaptable load balancing framework*, CASCON '03: Conference of the Centre for Advanced Studies on Collaborative research, 2003.

[PWGB10] Pandey S., Wu L., Guru S.M. and Buyya R., A Particle Swarm Optimization-Based Heuristic for Scheduling Workflow Applications in Cloud Computing Environments, AINA 2010, pp. 400-407, 2010.

[RaHa00] Rajagopalan A. and Hariri S., *An agent based dynamic load balancing system*, International Workshop on Autonomous Decentralized System, 2000.

[Rahm08] Rahman M. A., *Load balancing in DHT based P2P networks*, 5th Int. Conference on Electrical and Computer Engineering, ICECE, 2008.

[RFHK01] Ratnasamy S., Francis P., Handley M., and Karp R., *A scalable content-addressable network*, SIGCOMM, 2001.

[Roch98] Rocha, L. M., *Selected Self-Organization and the Semiotics of Evolutionary Systems*, Evolutionary Systems: The Biological and Epistemological Perspectives on Selection and Self- Organization, S. Salthe, G. Van de Vijver, and M. Delpos (eds.), Kluwer Academic Publishers, pp. 341-358, 1998.

[RoMe08] Roach C. and Menezes R., *Handling Dynamic Networks Using Evolution in Ant-Colony Optimization*, 21th Int. Conf. on Industrial, Engineering and Other Appl. of Applied Intelligent Systems, 2008.

[RoWo05] Rozansky N. and Woods E., Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives, Addison-Wesley Professional, 2005.

[SaCM99] Sancese S., Ciancarini P. and Messina A., *Message passing vs. tuple space coordination in an aerodynamics application*, 5th International Conference on Parallel Computing Technologies, LNCS, 1999.

[ScCr07] Schmickl T. and Crailsheim K., *A Navigation Algorithm for Swarm Robotics Inspired by Slime Mold Aggregation*, LNCS, Swarm Robotics, pp. 1-13, 2007.

[Schu02] Schumpeter J.A., The economy as a whole – seventh chapter of the theory of economic development. Industry and Innovation, vol. 9(1/2), 2002.

[Shal01] Shalizi C. R., Causal Architecture, *Complexity and Self-Organization in Time Series and Cellular Automata*, PhD thesis, University of Wisconsin-Madison, 2001.

[ShLe09] Shoham Y. and Leyton-Brown K., *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge University Press, 2009.

[Skie08] Skiena S.S., The Algorithm Design Manual, Springer, 2008.

[SLTD02] Sahin E., Labella T., Trianni V., Deneubourg J.-L., Rasse P., Floreano D., Gambardella L., Mondada F., Nolfi S., and Dorigo M., *SWARM-BOTS: Pattern Formation in a Swarm of Self-Assembling Mobile Robots*, Proceedings of the IEEE Int. Conference on Systems, Man and Cybernetics, 2002.

[ŠeCv02] Šešum V. and Cvetković D., *Genetic Algorithms for Internet Search: Examining the Sensitivity of Internet Search by Varying the Relevant Components of Genetic Algorithm*, International Conferences on Advances in Infrastructure for Electronic Business, Education, Science, Medicine, and Mobile Technologies on the Internet, 2002.

[ŠeKr99] Šešum V. and Kratica J., *Some mathematic methods of solving the geophysical inversion problem,* 25th Jupiter Conference, pp. 2.61-2.66, 1999.

[ŠeKT00] Šešum V., Kratica J. and Tošić D., *Solving the Geophysical Inversion Problem Using Genetic Algorithms* , YU Journal of Operational Research, vol. 10(2), pp. 283-292, 2000.

[ŠeKü08] Šešum-Cavic V. and Kühn E., *Instantiation of a Generic Model for Load Balancing with Intelligent Algorithms*, 3rd International Workshop on Self-Organizing Systems (IWSOS), LNCS, 2008.

[ŠeKü09] Šešum-Cavic V. and Kühn E., *Peer-to-Peer Overlay Network based on Swarm Intelligence*, Engineering Societies in the Agents World (ESAW), LNAI Springer Verlag, 2009.

[ŠeKü10a] Šešum-Cavic V. and Kühn E., *A Swarm Intelligence Appliance to the Construction of an Intelligent Peer-to-Peer Overlay Network*, International Conference on Complex, Intelligent and Software Intensive Systems, IEEE/CISIS/COCOSS, 2010.

[ŠeKü10b] Šešum-Cavic V. and Kühn E., *Applying swarm intelligence algorithms for dynamic load balancing to a Cloud Based Call Center*, 4th IEEE International Conference on Self-Adaptive and Self-Organizing Systems, IEEE/SASO, 2010.

[ŠeKü10c] Šešum-Cavic V. and Kühn E., *Comparing configurable parameters of swarm intelligence algorithms for dynamic load balancing*, International Workshop on-Self-Adaptive Network, IEEE/SASO/SAN, 2010.

[ŠeKü11] Šešum-Cavic V. and Kühn E., *Self-Organized Load Balancing through Swarm Intelligence*, to appear, Springer Verlag Studies in Computational Intelligence, book chapter.

[ŠeTo02] Šešum V. and Tošić D., *Genetic Algorithms and Smoothing Filters in Solving the Geophysical Inversion Problem*, YU Journal of Operational Research, vol. 12(2), pp. 215-226, 2002.

[ShKr94] Shivaratri N.G. and Krueger P., *Adaptive Location Policies for Global Scheduling*, IEEE Trans. on Software Engineering vol. 20, pp. 432-444, 1994.

[ShLe09] Shoham Y. and Leyton-Brown K., *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*, Cambridge University Press, 2009.

[ShSh03] Shalizi C.R. and Shalizi K.L., *Quantifying Self-Organization in Cyclic Cellular Automata*, Noise in Complex Systems and Stochastic Dynamics, Lutz Schimansky-Geier and Derek Abbott and Alexander Neiman and Christian Van den Broeck, SPIE, vol. 5114, 2003.

[Slug07] Sluga T.A., *Modern C++ implementation of the LINDA coordination language for distributed applications*, Technical report, 2007.

[SMKKB01] Stoica, I., Morris, R., Karger, D., Kaashoek, M., and Balakrishnan, H., *Chord: A scalable peer-to-peer lookup service for internet applications,* The Annual Conference of the Special Interest Group on Data Communication, ACM SIGCOMM, 2001.

[Solo88] Solomon H. Y., *Movement-produced invariants in haptic explorations: An example of a self-organizing information-driven, intentional system*, Human Movement Science, vol. 7, pp. 201-223, 1988.

[Stew01] Stewart M., *The Coevolving Organization*, Decomplexity Associates LtD, Rutland, UK, 2001.

[Stüt97] Stützle T., *MAX-MIN Ant System for the quadratic assignment problem*, Technical Report AIDA-97-4, FB Informatik, TU Darmstadt, Germany, 1997.

[TaDa00] Tarakanov A. and Dasgupta D., *A formal model of an artificial immune system*, BioSystem, vol. 55, pp. 151-158, 2000.

[TaVa02] Tanenbaum A.S., Van Steen M., *Distributed Systems: Principles and Paradigms*, Prentice Hall, 2002.

[TaVe05] Tang N. and Vemuri V.R., *An artificial immune system approach to document clustering*, SAC/ACM symposium on Applied Computing, 2005.

[TBSDL01] Theraulaz G., Bonabeau E., Sauwens C., Deneubourg J.L., Lioni A., Libert F., Passera L. and Solé R., *Model of droplet dynamics in the Argentine ant Linepithema humile (Mayr)*, Bulletin of Mathematical Biology, vol. 63, pp. 1079–1093, 2001.

[TiNK02] Timmis J., Neal M. and Knight T., *AINE: Machine Learning Inspired by the Immune System*, IEEE Transactions on Evolutionary Computation, 2002.

[ToVi02] Toth P. and Vigo D., *Models, relaxations and exact approaches for the capacitated vehicle routing problem,* Discrete Applied Mathematics, vol. 123, pp.487-512, 2002.

[TrTU06] Trumler W., Thiemann T. and Ungerer T., *An Artificial Hormone System for Self-organization of Networked Nodes*, Biologically Inspired Cooperative Computing, IFIP 19th World Computer Congress, TC 10: 1st IFIP International Conference on Biologically Inspired Computing, 2006.

[TSTNT05] Thant H., San K., Tun K., Naing T. and Thein N., *Mobile agents based load balancing method for parallel applications*, 6th Asia -Pacific Symposium on Information and Telecommunication Technologies (APSITT 2005), 2005.

[TyAB06] Tyrrell, A., Auer, G., and Bettstetter, C., *Fireflies as role models for synchronization in ad hoc networks*, 1st International Conference on Bio-inspired Models of Network, Information and Computing Systems (BIONETICS), 2006.

[TLYD05] Tian J., Liu Y., Yang X.-H and Du R., *Design and analysis of a novel load balancing model based on mobile agent*, Advances in Machine Learning and Cybernetics, 4th International Conference on Machine Learning and Cybernetics (ICMLC), 2005.

[Usyc07] Usychenko V., *Evolution of self-organizing systems from the standpoint of mechanics and thermodynamics*, Technical Physics, MAIK Nauka, vol. 52(7), 2007.

[VaBr01] Van Dyke Parunak H. and Brueckner S., *Entropy and self-organization in multi-agent systems*, 5th International Conference on Autonomous Agents, AGENTS '01, 2001.

[VaVS98] Van Steen M., Van der Zijden S. and Sips H.J., *Software Engineering for Scalable Distributed Applications*, 22nd Computer Software and Applications Conference, 1998.

[ViCa09] Viroli M. and Casadei M., *Biochemical Tuple Spaces for Self-organising Coordination Coordination Models and Languages*, LNCS vol. 5521, pp. 143-162, 2009.

[ViCO09] Viroli M., Casadei M. and Omicini A., *A Framework for Modeling and Implementing Self-Organizing Coordination*, 24th Annual ACM Symposium on Applied Computing, (SAC 2009), 2009.

[ViCG07] Viroli M., Casadei M. and Gardelli L., *A Self-Organizing Solution to the Collective Sort Problem in Distributed Tuple Spaces*, SAC '07: ACM Symposium on Applied Computing, 2007.

[ViSk02] Vittikh V.A. and Skobelev P. O., Multi-agent systems for modelling of self-organization and cooperation processes, 13th International Conference on the Application of Artificial Intelligence in Engineering, pp. 91–96, 2002.

[VoBW08] Von Renteln A., Brinkschulte U. and Weiss M., *Examinating Task Distribution by an Artificial Hormone System Based Middleware,* 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing, (ISORC), 2008.

[vonF67] von Frisch K*., The dance language and orientation of bees*, Harvard University Press, Cambridge, MA, 1967.

[Vygo78] Vygotsky L.S., Mind and society: The development of higher mental processes, Harvard University Press, Cambridge, 1978.

[WaLi03] Wang Y. and Liu J., *Macroscopic model of agent-based load balancing on grids*, 2nd International Joint Conference on Autonomous Agents and Multiagents systems, AAMAS '03, 2003.

[WaRC00] Waldman, M., Rubin A.D. and Cranor L.F., *Publius: A robust, tamper-evident, censorship-resistant web publishing system*, 9th USENIX Security Symposium, 2000.

[WaYW04] Wang S.C., Yan K.Q. and Wei C.H., Mobile target dvertising by combining self-organization map and decision tree, IEEE International Conference on e-Technology, e-Commerce and e-Service, pp. 249–252, 2004.

[Weis80] Weiss S.F., *A probabilistic algorithm for nearest neighbour searching*, 3rd Annual ACM Conference on Research and Development in Information Retrieval (SIGIR '80), pp. 325-333, 1980.

[Werb00] Werbach K., Syndication: The emerging model for business in the internet era, Harvard Business Review, vol.85, pp. 85–93, 2000.

[Wien61] Wiener N., *Cybernetics: or Control and Communication in the Animal and the Machine*, MIT Press, 1961.

[WMHZ02] Weikum G., Monkeberg A., Hasse C. and Zabback P., *Self-tuning Database Technology and Information Services: from Wishful Thinking to Viable Engineering*, Proceedings of the 28th Very Large Databases Conference (VLDB), pp. 20–31, 2002.

[WoLC08] Wong L.P, Low M.Y.H. and Chong C.S, *A Bee Colony Optimization for Travelling Salesman Problem*, 2nd Asia International Conference on Modelling and Simulation, 2008.

[WMLF98] Wyckoff P., McLaughry S.W., Lehman T.J. and Ford D.A., *T-Spaces,* IBM Systems Journal, vol. 37(3), pp. 454–474, 1998.

[WTPWN05] Werner-Allen, G., Tewari, G., Patel, A., Welsh, M., and Nagpal, R., *Firefly-inspired sensor network synchronicity with realistic radio effects*, 3rd International Conference on Embedded Networked Sensor Systems, 2005.

[Yang08] Yang X-S, *Nature-Inspired Metaheuristic Algorithms*, Luniver Press, 2008.

[Yu08] Yu H., *Optimizing task schedules using an artificial immune system approach*, 10th Annual Conference on Genetic and Evolutionary Computation, 2008.

[ZhHu05] Zhu Y. and Hu Y., *Efficient, proximity-aware load balancing for DHT-based P2P systems,* IEEE Trans. on Parallel and Distributed Systems, vol.16(4), pp.349-361, 2005.

[ZhKJ01] Zhao, B., Kubiatowicz, J., and Joseph, A., *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*, Tech. Rep. UCB/CSD-01-1141, Computer Science Division, University of California, Berkeley, 94720, 2001.

[ZhLC06] Zhang J., Lo W.L. and Chung H., *Pseudocoevolutionary Genetic Algorithms for Power Electronic Circuits Optimization,* IEEE Trans. Systems, Man and Cybernetics, vol. 36(4), 2006.

[Zhou88] Zhou S., *A trace-driven simulation study of dynamic load balancing*, IEEE Trans. on Software Engineering, vol. 14, pp.1327-1341, 1988.

[ZoDK07] Zoels S., Despotovic Z. and Kellerer W., *Load balancing in a hierarchical DHT-based P2P system*, International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom, 2007.

[ZoJM01] Zoethout K., Jager W. and Molleman E., *Self-organizing processes of task allocation*, 5th Simulating Societies Conference, 2001.

[ZoTe01] Zomaya A.Z. and Teh Y.H., *Observations on using genetic algorithms for dynamic load-balancing*, IEEE Trans. on Parallel and Distributed Systems, vol. 12(9), pp. 899-911, 2001.

[XuBh06] Xu Z.and Bhuyan L., *Effective load balancing in P2P systems*, 6th IEEE International Symposium on Cluster Computing and the Grid, CCGRID '06, 2006.

[XuDH09] Xu H., Ding Y., and Hu Z., *Adaptive immune genetic algorithm for logic circuit design*, 1st Summit on Genetic and Evolutionary Computation, China, 2009.

[XuGu07] Xu M. and Guan J., *Routing based load balancing for unstructured P2P networks*, Future Generation Communication and Networking, vol. 2, pp.332-337, 2007.

URL REFERENCES

[ACloud11] Amazon Elastic Compute Cloud http://aws.amazon.com/ec2/, last accessed: January, 2011.

[Gnut03] Gnutella 2003, The Gnutella web site: http://gnutella.wego.com.

[Godl97] Golden R., Self-Organizing Systems: a resource for teachers 1997, http://sciphilos.info/docs_pages/docs_Golden_sos_css.html

[JXTA10] http://www.sun.com/sotware/jxta, last accessed: 2010.

[Kaza03] Kazaa 2003, The Kazaa web site. http://www.kazaa.com.

[Mozart11] MozartSpaces http://www.mozartspaces.org/1.0-alpha/, last accessed: January, 2011.

[PyLi10] PyLinda. Pylinda project homepage. http://code.google.com/p/pylinda/

[Sutc90] Sutcliffe G., Prolog-D-Linda v2: A New Embedding of Linda in SICStus Prolog, http://www.cs.miami.edu/~geoff/Papers/Other/1993_Sut93_TR-JCU-CS-93-6.pdf, 1990.

[TripC08] Triple Space Communication, Semantic Clustering and Self-Organization in Triple Space, http://www.tripcom.net/docs/del/D2.4.pdf, 2008.

[XVSM11] eXtended Virtual Shared Memory http://www.xvsm.org, last accessed: January, 2011.