Informatics

A Framework for Self-Initiative Peer Clustering Agents

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Laura Fagagnini
Matrikelnummer 01302472

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eva Kühn
Mitwirkung: Dipl.-Math. Dr.techn. Vesna Šešum-Čavić

Wien, 7. Februar 2021

Laura Fagagnini   Eva Kühn

Informatics

A Framework for Self-Initiative Peer Clustering Agents

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Software Engineering & Internet Computing

by

Laura Fagagnini
Registration Number 01302472

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eva Kühn
Assistance: Dipl.-Math. Dr.techn. Vesna Šešum-Čavić

Vienna, 7th February, 2021

Laura Fagagnini	Eva Kühn

Erklärung zur Verfassung der Arbeit

Laura Fagagnini
Bachgasse 2/17, 2013 Göllersdorf

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 7. Februar 2021

Laura Fagagnini

Acknowledgements

First of all, I would like to thank my supervisors Eva Kühn and Vesna Šešum-Čavić, who gave me the opportunity to work on this interesting topic. During the course of this thesis, their constructive feedback and guidance was of great value.

Furthermore, I would like to thank Stefan Craß for helping me to understand the Peer Model Java implementation, such that the SIPCA framework would fit in.

Most importantly, I would like to thank my family and friends for their never ending support and understanding. I am deeply grateful for their inspiration and motivation.

Abstract

In most Peer-to-Peer networks, peers are placed randomly or based on their geographical position. This can lead to a performance bottleneck and thus, performance in such networks can be extremely poor. This problem can be solved by using peer clustering algorithms, aiming at grouping peers, which have certain characteristics in common, together as neighbors.

This thesis provides a benchmarking framework, which allows the systematic benchmarking, evaluation and comparison of peer clustering algorithms. The resulting application is based on the Peer Model, a coordination-based programming model. Furthermore, the framework provides extensive configuration possibilities and easy exchangeability of the applied peer clustering algorithms.

As additional contributions, two innovative algorithms, Slime Mold and Slime Mold K-Means, based on the life-cycle of the Dictyostelium discoideum slime mold are implemented. Those two algorithms are presented and described in detail in Chapter 5. They are competitively benchmarked, evaluated and compared to the following nine well-known conventional and swarm-based algorithms: Artificial Bee Colony, Artificial Bee Colony combined with K-Means, Ant-based Clustering, Ant K-Means, Fuzzy C-Means, Genetic K-Means, Hierarchical Clustering, K-Means and Particle Swarm Optimization.
Slime Mold and Slime Mold K-Means outperform all other swarm-inspired algorithms in terms of execution time and quality of the clustering solution. Although they are usually midranged, they never provide unwanted massive increase or decrease in clustering effectiveness results, unlike other peer clustering algorithms.

Kurzfassung

In den meisten Peer-to-Peer Netzwerken werden Peers zufällig oder anhand ihrer geografischen Position verbunden. Das kann zu einem Leistungsengpass führen, worunter die Performanz eines solchen Netzwerkes leiden kann. Dieses Problem kann durch die Nutzung von Peer Clustering Algorithmen, welche die Peers anhand bestimmter Charakteristika gruppieren, gelöst werden.

Diese Diplomarbeit liefert ein Framework, das systematische Benchmark-Tests von Peer Clustering Algorithmen, deren Auswertung und Vergleich ermöglicht. Die resultierende Applikation basiert auf dem Peer Model, einem koordinationsbasierten Programmiermodell. Außerdem bietet das Framework vielfältige Konfigurationsmöglichkeiten und erlaubt die einfache Austauschbarkeit der verwendeten Peer Clustering Algorithmen.

Als weiterer Beitrag werden zwei innovative Algorithmen, Slime Mold und Slime Mold K-Means, die auf dem Lebenszyklus des Dictyostelium discoideum Schleimpilzes basieren, implementiert. Die beiden Algorithmen werden in Kapitel 5 vorgestellt und detailliert beschrieben. Diese werden kompetitiv gemessen, evaluiert und mit den neun folgenden, bekannten konventionellen und schwarm-basierten Algorithmen verglichen: Artificial Bee Colony, Artificial Bee Colony combined with K-Means, Ant-based Clustering, Ant K-Means, Fuzzy C-Means, Genetic K-Means, Hierarchical Clustering, K-Means and Particle Swarm Optimization.
Slime Mold und Slime Mold K-Means übertreffen sämtliche andere schwarm-basierten Algorithmen in Bezug auf Ausführungszeit und Qualität der Clustering Lösung. Denn obwohl sie sich meist im Mittelfeld befinden, zeigen sie, im Gegensatz zu anderen Algorithmen, keine extremen, unerwünschten Schwankungen bezüglich der Clustering Effektivität.

Contents

1 Introduction **1**
1.1 Problem Statement 1
1.2 Aim of the Work 2
1.3 Methodological Approach 3
1.4 Structure of the Thesis 3

2 Technical Background & Related Work **5**
2.1 P2P Networks 5
2.2 Peer Model 7
2.3 Clustering Algorithms 8
2.4 Related Work 12
2.5 Summary 13

3 Framework Description **15**
3.1 Local Peer Pattern 15
3.2 Pattern Composition 17
3.3 Framework Composition 19
3.4 Core Framework Components 23
3.5 Additional Framework Component 40
3.6 Summary 45

4 Implementation Details **47**
4.1 Peer Model Implementation & Extensions 47
4.2 Services 48
4.3 Interfaces and Classes 52
4.4 Metrics 56
4.5 Framework Execution 59
4.6 Framework Configuration 59
4.7 Summary 62

5 Swarm-Inspired Algorithms for Peer Clustering **63**
5.1 P2P Resource 63
5.2 Clustering Criteria 64

5.3 Artificial Bee Colony (ABC) 68
5.4 Combination of Artificial Bee Colony Algorithm and K-Means (ABCK) 71
5.5 Ant-based Clustering 72
5.6 Ant K-Means 74
5.7 Particle Swarm Optimization (PSO) 76
5.8 Slime Mold 77
5.9 Slime Mold K-Means 80
5.10 Summary 81

6 Evaluation **83**
6.1 Benchmark Methodology 83
6.2 Sensitivity Analysis 84
6.3 Raw Result Data 93
6.4 Competitive Analysis 100
6.5 Statistical Significance of the Results 111
6.6 Summary 115

7 Conclusion and Future Work **117**
7.1 Future Work 117
7.2 Conclusion 118

List of Figures **121**

List of Tables **123**

List of Algorithms **125**

Acronyms **127**

Bibliography **129**

Web-References **133**

CHAPTER 1

Introduction

This Chapter provides an introduction to the thesis and discusses the problem statement. Additionally, the expected results and the methodological approach to accomplish these results are clarified. Finally, this Chapter provides the structure of the thesis.

1.1 Problem Statement

Peer-to-peer (P2P) networks have evolved over the recent years and thus, have been an interesting field in the research area of distributed systems [10]. In the internet there exist many popular file-sharing applications like Gnutella [12] or Napster [12], which were designed as P2P networks. Established patterns like load balancing [40] or load clustering [29] can be used in such a system when suffering from high loads. Unfortunately, these patterns are not ideal, if the positioning of the peers is the performance bottleneck. This is not unusual, due to the fact that in most P2P networks peers are placed randomly or based on their geographical position [35]. Consequently, the performance in such networks can be extremely poor.

The purpose of *clustering* is to group a collection of observations into clusters. Thus, somehow similar elements belong to the same cluster, whereas dissimilar elements belong to another cluster [29].
As the name already indicates, *peer clustering* aims at grouping peers, which have certain characteristics in common, together as neighbors [41].
Due to the fact that peers are leaving and entering the network dynamically, also peer clustering has to be a dynamic procedure. With peer clustering query performance can be significantly improved compared to a random network topology. This is the case as requests are routed more efficiently and only to nodes which are likely to fit the request. Besides, if a cluster containing a node, which is likely to fit the request, can be found, query flooding through the whole network is not necessary. Consequently, this means that workload on nodes, which are probably not fitting the request, can be reduced [41, 10].

Furthermore, on top of peer clustering load balancing and load clustering can be implemented, respectively. Thus, load can be shifted within a cluster of peers, such that it is not necessary to scan the whole network for an under-loaded node.
Additionally, workload can be sent directly to an appropriate cluster. Hence, when the required computational power for a task is known, the workload will not be forwarded to a peer having far more computational power than required to avoid resource wastage. Also, it should not be forwarded to a node which has far less computational power than required, as this either is time consuming or the task even has to be shifted to a node having more resources available.

The mentioned benefits emphasize the importance of peer clustering in P2P networks. But there exist many different algorithms and approaches for peer clustering (cf. section 2.3 and section 2.4).
Thus, finding the best suiting algorithm for a specific problem can be a time-consuming process. This fact encourages the need for a general framework to compare different peer clustering algorithms, facilitating the search for the most suitable one.
Unfortunately, no such general pattern to describe peer clustering was found. As a consequence, for each network peer clustering has to be implemented individually and currently, for this problem does not exist a general solution.
Thus, the main goal of the thesis is to describe and implement a generic pattern for peer clustering.

1.2 Aim of the Work

The main expected result of this work is to develop a benchmarking framework for unstructured P2P networks, called *SIPCA*, standing for "Self-Initiative Peer Clustering Agents", in the course of this thesis. This involves not only a theoretical description of the framework, but especially its actual implementation. The framework shall allow the plugging of different peer clustering algorithms, such that easy exchangeability of the applied algorithms is possible, and enable systematic benchmarking and comparison of these algorithms. Furthermore, the framework shall be problem independent, as it should be used to find the best suiting algorithm for a specific problem. Moreover, the implementation shall be based on the Peer Model [27], a coordination-based programming model, which is presented in more detail in section 2.2.

For the benchmarking process different algorithms will be applied, i.e. for this framework a set of clustering algorithms is going to be implemented (cf. section 2.3). This includes conventional algorithms, like K-Means [21], as well as intelligent algorithms, such as Ant K-Means [30]. The evaluation part at the end of this work provides a comparison of these clustering algorithms and is expected to show which algorithms perform particularly well in certain scenarios.

1.3 Methodological Approach

To achieve the expected results the following methodologies are applied:

1. **Literature Review:** The main goal of this step is to gather information in respect to the technical background of the topic, related work and state-of-the art.

2. **Implementation:** In this step the modeling and implementation of the framework takes place. For this purpose, firstly, the environment of the framework is implemented, such that the clustering algorithms can be implemented in a Plug & Play manner. Then, the clustering algorithms for benchmarking are implemented.

3. **Evaluation:** The evaluation of the clustering algorithms includes the benchmarking of different algorithms. Thus, the algorithms, including fine tuning of parameters, are compared in terms of performance, i.e. absolute execution time, and effectiveness.

1.4 Structure of the Thesis

This thesis is structured as follows:
In Chapter 2 the technical background of the thesis and the related work are discussed. This includes a short overview over the implemented algorithms.
Chapter 3 covers the general structure of the benchmarking framework, including its architecture, its pattern composition and its components, whereas Chapter 4 deals with the actual implementation, the framework configuration and its execution.
In Chapter 5 the implemented swarm-inspired peer clustering algorithms are discussed thoroughly. In Chapter 6 the benchmark methodology is defined and the benchmark results are evaluated. Finally, Chapter 7 discusses possible future improvements to the proposed framework.

CHAPTER 2

Technical Background & Related Work

This Chapter contains the the description of the technical background of the thesis and its related work. It is divided into four parts.
First of all, general information about P2P networks is provided. Next, the Peer Model is described. Afterwards, the implemented peer clustering algorithms are shortly discussed. Finally, related state-of-the-art approaches are presented.

2.1 P2P Networks

Overall, it can be said there exist various different definitions of peer-to-peer, as it is just pointed out by Androutsellis-Theotokis and Spinellis [1]. Therefore, they propose their own definition as follows:

> *"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."*
> [1, p. 337]

This definition leads to the following characteristics which make a distributed system a peer-to-peer system. One of the most frequently mentioned characteristics is the ability to share computer resources directly while avoiding the usage of a central server for the mediation between individual nodes. Furthermore, P2P systems have the capability of self-organization and fault-tolerance. This also implies the demand for an adaptive

network topology due to churn, the fact that nodes may enter or leave the network anytime [1, 6].
Additionally, Buford and Yu [6] state the characteristic of symmetry, meaning all participants in a P2P network take equivalent roles such that every node acts as client and server at the same time.
However, both, Androutsellis-Theotokis and Spinellis [1] and Buford and Yu [6], point out, that not every P2P system includes every mentioned characteristic, especially the symmetry of peers as many P2P systems are using super peers or other special peer roles.

P2P systems play an important role in research in the field of distributed system and have become a notable technology for collecting, publishing, distributing and sharing data, especially because of their high scalability, their low entry barrier and the fact that they are applicable for different application types [6].

2.1.1 P2P Overlay Networks

The network of a P2P system is an overlay network because a virtualized network is formed over the physical network [6].
There exist different approaches to classify a P2P overlay network. In [1] they are distinguished by centralization and structure, respectively. They declare the following three categories to describe the degree of centralization:
In a *Purely Decentralized Architecture* full symmetry of roles is given, meaning that every node acts as both, server and client, and there exists no central coordination.
The basis of *Partially Centralized Architectures* is the same as of purely decentralized architectures. But some of the nodes fulfill a more important role, supporting coordination and collaboration within the network. As these supernodes are assigned and replaced dynamically, they are not seen as single points of failure.
In a *Hybrid Decentralized Architecture* the interaction between peers is facilitated by a central server by maintaining metadata and performing lookups. Consequently, this central server is a single point of failure. This also implies, that these systems are not as scalable as the other architectures mentioned above.

Classification by the means of network structure is distinguished by Androutsellis-Theotokis and Spinellis [1] the following way:
In an *Unstructured* network a peer, which joins the network, has no knowledge of the network's topology [33]. The placement of peers, and thus also content, is often random. This means, typically, content has to be located. Therefore, many searching algorithms exist, e.g. flooding the network with queries in a breadth-first or depth-first manner. Popular examples for unstructured P2P networks are Napster or Gnutella.
In contrast, in a *Structured* network it is specified where a file shall be located. For this purpose a distributed routing table is provided, to map the content to its location. Thus, queries can be routed more efficiently.

2.2 Peer Model

Kühn et al. presented the *Peer Model*, a coordination-based programming model, in [27] and [28]. It is a space-based middleware meant to close the gap between design and implementation. Due to the asynchronous, blackboard-based communication, enabled by a virtually shared memory, a high level of decoupling is provided by the Peer Model. This ensures that the autonomy of peers is maintained [27, 28].

The main component of the Peer Model is a *peer*, existing in a Peer Space.
Every peer has a unique name (URI) and two containers: an input space *peer-in-container* (PIC) and an output space *peer-out-container* (POC). A peer receives request entries via its PIC, takes them out to process them, and then puts the replies into its POC. Consequently, inter-peer collaboration takes place between a peer's POC and the PIC of a foreign peer [27].

Peers are nested entities, meaning a peer, a so called *sub-peer*, can be created within the scope of another peer. Furthermore, there exist *space peers*, specialized peers having the functionality of a space container, such that its PIC and POC are melted and all entries are stored in one place. It can be used to store data for the purpose of sharing it between concurrent processes [27].

Wirings are contained in a peer and serve for the transportation of entries between PIC and POC containers. They are the system's only active part. A wiring consists of the three following parts: *guards*, *service calls* and *actions* [27].
Guards represent conditions and use space operations to receive entries from a container. A service is called if all guards are fulfilled. Therefore, the received entries are used as input parameters and the service possibly produces result entries. In the end, actions are executed. They deliver the resulting entries to a container using a write operation [28].

According to [44], for the movement of entries the following space operations, which are used by the guards, are available: read, take, delete, none and test.
Using the *read* and *take* operation, respectively, one or more entries are received. Whilst the take operation removes the entry from the PIC container of a peer, the read operation does not. This means the take operation is a consuming get operation, whereas the read operation is a non-consuming get operation. Also the usage of the *delete* operation removes an entry from the PIC, but in this case it is not passed to the service.
The *none* operation ensures the non-existence of an entry of a specific type, whereas with the *test* operation the availability of an entry with a specific type can be verified. Both of these operations do not provide an entry to a service.

Furthermore, the subsequent entry properties are provided:
If an entry has a time-to-start (TTS) property, a guard must not take it until the TTS is reached.
An entry with a *time-to-live* (TTL) property has a defined life-time. When the TTL expires, the entry is wrapped in a timeout exception [28].

2.2.1 Peer Model Notation

For the purpose of illustration an example instance of a peer is shown in figure 2.1. The graphical notation, which is used in the thesis, to depict peers and other components of the Peer Model is based on [9].

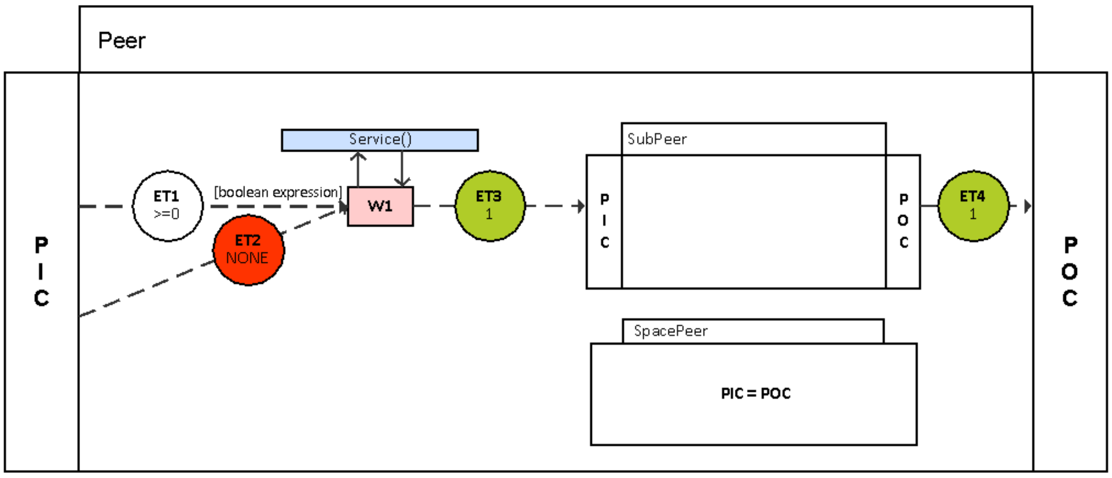

Figure 2.1: Example for a peer, including a sub-peer, a space peer and a wiring.

A peer, with its PIC and POC, is represented by rectangles. Sub-peers are contained within peers and have the same notation. Pink colored squares illustrate wirings, whereas a service called by a wiring is visualized by a blue shaded square. The incoming arrows of a wiring represent its guards and the outgoing ones embody its actions. A guard specific query is depicted by a boolean expression in square brackets.
Entries are embodied by circles, where in the upper half of the circle the entry type is stated. In case of a guard, the lower half of the circle declare the link count. In case of an action, it specifies how many entries of this type are produced by the wiring. The coloring of the entries represent the link operation: white is the read operation, green is the take operation, red is the none operation and yellow is the test operation.

2.3 Clustering Algorithms

There exist many different algorithms coping with the problem of clustering. This includes conventional, as well as swarm-inspired algorithms. Therefore, the following eleven clustering algorithms are used for the benchmarks and discussed in the following section.

2.3.1 Conventional Algorithms

In this section three of the most widely used and well known conventional clustering algorithms are presented: Hierarchical Clustering, K-Means and Fuzzy C-Means.

2.3.1.1 Hierarchical Clustering

Hierarchical Clustering builds a hierarchy of clusters, which can be done basically in two ways: **Agglomerative** [16] and **Divisive** [17]. The first approach creates a cluster for each element to be clustered. Then, these clusters are merged together iteratively until the expected number of clusters is reached. This procedure is chosen for the implementation and therefore, described in more detail below. The second one starts with having all elements in one cluster, which is iteratively split until the expected number of clusters is reached.

The Agglomerative Hierarchical Clustering algorithm proceeds the following way:

1. Start with m clusters, each containing one element, where m is the number of elements to be clustered

2. Calculate the distance of all clusters

3. Merge the two closest clusters

4. Recalculate the cluster mean

5. Repeat 2. - 4. until the expected number of clusters is reached

2.3.1.2 K-Means

According to Jain et. al. [21] and Tan et. al. [43], the K-Means algorithm is one of the simplest and commonly used clustering algorithms. Firstly, k centroids are created, where k is the expected number of clusters. Then, each element to be clustered is associated with the cluster containing the nearest centroid. From this association, in the next step, k new centroids are calculated. The step of reassigning the elements and recalculating the centroids is repeated, until the centroids do not move anymore.

In detail, the K-Means algorithm proceeds as follows:

1. Create k clusters, populated with one random element as representative (i.e. centroid)

2. Compute the distance between each element and each of the centroids

3. Assign each element to the cluster with the nearest centroid

4. Recalculate the centroid of each cluster (i.e. cluster mean)

5. Repeat 2. - 4. until the centroids do not change anymore

2.3.1.3 Fuzzy C-Means

The Fuzzy C-Means [13, 3, 4] algorithm is similar to the K-Means algorithm. The main difference lies in the fact that an element is allowed to belong to more than one cluster. Therefore, the degree of membership has to be calculated as a measure for choosing the best suiting clusters.

The Fuzzy C-Means algorithm proceeds the following way:

1. Create c clusters, populated with one random element as representative (i.e. centroid)

2. Compute the distance d_{ij} between each element and each of the centroids

3. Calculate the degree of membership u_{ij} of an element i to a certain cluster j, using the fuzzifier m

$$u_{ij} = \frac{1}{\sum_{r=1}^{c}(\frac{d_{ij}}{d_{ir}})^{\frac{2}{m-1}}}; 1 \leq i \leq N; 1 \leq j \leq c; m \geq 1 \tag{2.1}$$

4. Assign an element to all clusters, where the degree of membership exceeds a certain value

5. Recalculate the centroid of each cluster (i.e. cluster mean)

6. Repeat 2. - 5. until the centroids do not change anymore

2.3.2 Swarm-inspired Algorithms

Swarm-based algorithms are inspired by nature and aim at imitating the behavior of life forms organized in swarms, such as ant or bee colonies, and apply it to the solution of problems. Due to the fact that such colonies are also perceived to be decentralized and self-organized systems, similar to peer-to-peer systems, they are likely to provide satisfying results [19].

2.3.2.1 Artificial Bee Colony (ABC)

The Artificial Bee Colony (ABC) algorithm [22] is based on the foraging behavior of honey bees and each food source embodies a possible solution. Each food source has a certain amount of nectar representing the quality or fitness, respectively, of a food source. The bees check the quality of the food sources in order to find the best one and thus, also the best solution.

2.3.2.2 Combination of Artificial Bee Colony Algorithm and K-Means (ABCK)

The ABCK algorithm [2] is a combination of the Artificial Bee Algorithm (cf. 2.3.2.1) and classical K-Means (cf. 2.3.1.2). Thus, the procedure is basically the same as the procedure of ABC, only differing in the fact that each solution of ABCK is locally optimized by the execution of K-Means.

2.3.2.3 Ant-Based Clustering

Ant-Based Clustering [34] is inspired by the behavior of ants. The ants act on a two-dimensional grid which is populated randomly with the items to be clustered. An ant is empowered to execute one of two actions: An unloaded ant can pick up an element lying on a field currently visited by the ant. An ant carrying an element can drop it on a free cell. These actions are decisions influenced by the ants perception of the environment.

2.3.2.4 Ant K-Means

The Ant K-Means algorithm, proposed by Kuo et. al. [30], is based on combining conventional K-Means clustering with ant colony optimization. Ant colonies leave a trail of pheromones, which is used by real ants to communicate with each other. Pheromones are left by an ant following a certain path. The more ants take this trail, the more pheromones lay on the trail. Consequently, this trail becomes more attractive for other ants and the shortest route can be obtained.
Thus, the conventional K-Means is modified in such a way, that the elements to be clustered are located in a cluster using a probability modified by the emitted pheromones.

2.3.2.5 Genetic K-Means

The Genetic K-Means algorithm [26] is a combination of classical K-Means and an algorithm based on the theory of evolution. Therefore, for the algorithm an initial population of clusters is used which gets evolved over several generations. While in evolutionary algorithms two genetic operations, namely mutation and crossover, are used, in Genetic K-Means the crossover operator is replaced by the K-Means operator, i.e. a one-step K-Means algorithm.
Until the maximum number of generations is reached, the following steps are performed over the population:

1. Selection
2. Mutation
3. K-Means Operator

2.3.2.6 Particle Swarm Optimization (PSO)

In Particle Swarm Optimization [37] an individual grouped into a swarm is referred to as a particle. A swarm can be seen as a flock of birds flying towards an optimum. Therefore, a particle searches for the best solution, using the best position encountered by itself and its swarm. Thus, a wide area can be searched while heading towards an optimum.

2.4 Related Work

The target area of the state-of-the-art are general frameworks for peer clustering in P2P networks. In the following section different approaches for clustering peers are discussed.

In [10] a Semantic Overlay Network (SON) is created. For this purpose peers with similar contents, like music genre, are connected to each other and consequently, building a semantic cluster. Therefore, the peers, the queries and the documents themselves, respectively, have to be classified in order to determine to which cluster(s) they belong. Thus, queries can directly be routed to the respective SON, actually improving the search performance.

Khambatti, Ryu and Dasgupta [25] propose a model for forming groups of peers implicitly, called communities, based on common interests. Those communities are formed as peers claim their interests analogous to social networks and are possibly overlapping. A peer's interests can be provided explicitly by the peer or implicitly identified from past queries. Furthermore they present a search technique which is based on the ability of the peers to form communities.
In [35] a similar model like in [25] is introduced, but here for the interests a predefined ontology is used instead of letting the peers claim their interests freely. This makes it easier for a peer to find other peers with similar interests.
In [32] a peer's interests are found out by extracting keywords from text documents in their storage. The keyword extraction affects the global keyword vocabulary and is treated as a decision problem. Although this is an interesting approach, it is only applicable for text documents.

In [41] two algorithms are proposed. They aim at creating an overlay network by clustering similar peers and are based on Schelling's model. One unique property is used to cluster peers having this property in common and thus, do not overlap. Schelling's model explains the existence of segregated neighborhoods in America. It consists of a 2-dimensional grid where two thirds of the cells are randomly populated with blue and red turtles. The turtles wish to have at least a certain percentage of neighbors having the same color as themselves. If this is not the case, the turtle moves to an adjacent cell. This continues until all turtles are satisfied with their neighbors.

To sum up, there exist many different approaches for peer clustering in P2P networks. Despite extensive literature review, no general framework using or even benchmarking different peer clustering algorithms could have been found.

2.5 Summary

In this Chapter the technical background and the related work of this thesis are discussed. The technical background is focused on P2P networks. Then, the Peer Model, a coordination-based programming model is introduced. It closes the gap between design and implementation and is, due to its high-level decoupling, perfectly suitable for the implementation of a generic benchmarking framework for peer clustering algorithms. Afterwards, the eleven implemented peer clustering algorithms are shortly presented. This includes conventional as well as swarm-inspired algorithms.

As no general framework, which uses or even benchmarks different peer clustering algorithms, was found, the related work is focused primarily on different approaches for peer clustering.

CHAPTER 3

Framework Description

In this Chapter the general structure of the framework is discussed. This includes the pattern used for a single peer, the pattern composition and the framework composition. Hence, it explains how the different components of the SIPCA framework cooperate. The chapter is structured as follows.
First of all, the pattern for a single node in the SIPCA framework is described. Afterwards, the pattern composition, i.e. the inter-node communication, is discussed. Then, the description of the framework composition follows. This includes the synchronization pattern, clustering trigger policies and the statistics pattern. Finally, the framework's core and additional components, including their wirings, are introduced.

3.1 Local Peer Pattern

A node in a P2P network has no knowledge about the actual network topology, but only about its current neighbors. For this purpose it has to store information about its current neighbors. Furthermore, a node needs to provide information about itself, such as technical characteristics or interests. All this information is required for the peer clustering algorithms.

Figure 3.1 shows how the described functionality is mapped to the Peer Model domain: A node in a P2P network is represented by a Peer in the Peer Model. In the SIPCA framework there exist four different types of peers:

- *Client Peer*, a peer making requests to the network
- *Coordination Peer*, a peer coordinating requests and clustering
- *Storage Peer*, a peer storing data and processing data queries

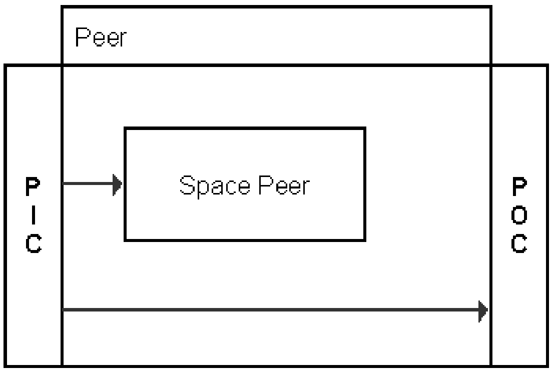

Figure 3.1: Local Peer Pattern.

- *Worker Peer*, a peer accepting and performing job requests like computations, elaborated in more detail in section 3.1.1

So each of the peers has a space peer as sub-peer to store before mentioned information. The Storage Peer also keeps its stored data there. The arrows indicate intra-node collaboration. How the peers communicate with each other, i.e. the pattern composition, is described in detail in section 3.2.

3.1.1 Worker Peer Pattern

While the inner structure of all peers in the framework is rather similar, the Worker Peer slightly differs in its components.

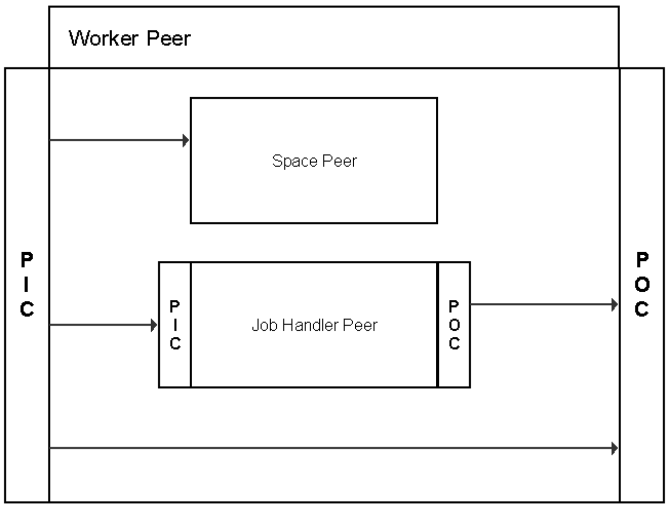

Figure 3.2: Local Worker Peer Pattern.

As shown in figure 3.1, in contrast to the other peers of the framework, the Worker Peer has an additional sub-peer, called Job Handler Peer. This sub-peer is responsible for actually executing an accepted job, while enabling the Worker Peer itself to handle further incoming job requests.

3.2 Pattern Composition

Actually, the inter-node communication is rather simple.

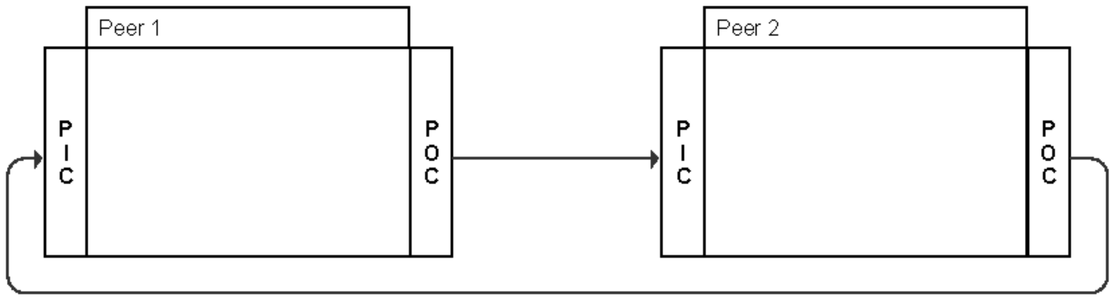

Figure 3.3: Basic Pattern Composition.

Figure 3.3 illustrates two single peers communicating with each other. As mentioned in section 2.2, this happens by moving an entry from one peer's POC to the PIC of another peer.

The composition of the described local peer patterns (cf. section 3.1) is used to establish the peer clustering pattern for the SIPCA framework. As there exist different types of peers in the framework, the actual pattern composition is more complex than the basic pattern composition described in figure 3.3. Additionally, at this point it is important to stress that although the framework strictly distinguishes between Storage and Worker Peers, they are both referred to as Node Peers as their external interaction is exactly the same.
While the number of Client Peers and Node Peers is not limited, there exists exactly one Coordination Peer per SIPCA Peer Space instance. A benchmark is run locally, when all peers of a benchmark exists in the same Peer Space. In this case, there exists only one Coordination Peer for the whole benchmark.

In figure 3.4 can be seen that, although the concept of communication between peers basically stays the same, it is more sophisticated in the SIPCA framework, as there are different types of peers communicating with each other for several reasons.
In a locally run benchmark, the Coordination Peer has two main tasks, a) forwarding requests, and b) executing the peer clustering algorithm and distributing the results. Thus, a Client Peer sends a request always to the Coordination Peer, which forwards it to a Node Peer belonging to the corresponding cluster, as depicted in figure 3.5. The Node Peer then decides upon certain criteria if a request can be fulfilled or not. If a Node Peer is able to fulfill the request, the request is accepted and processed. Afterwards, the

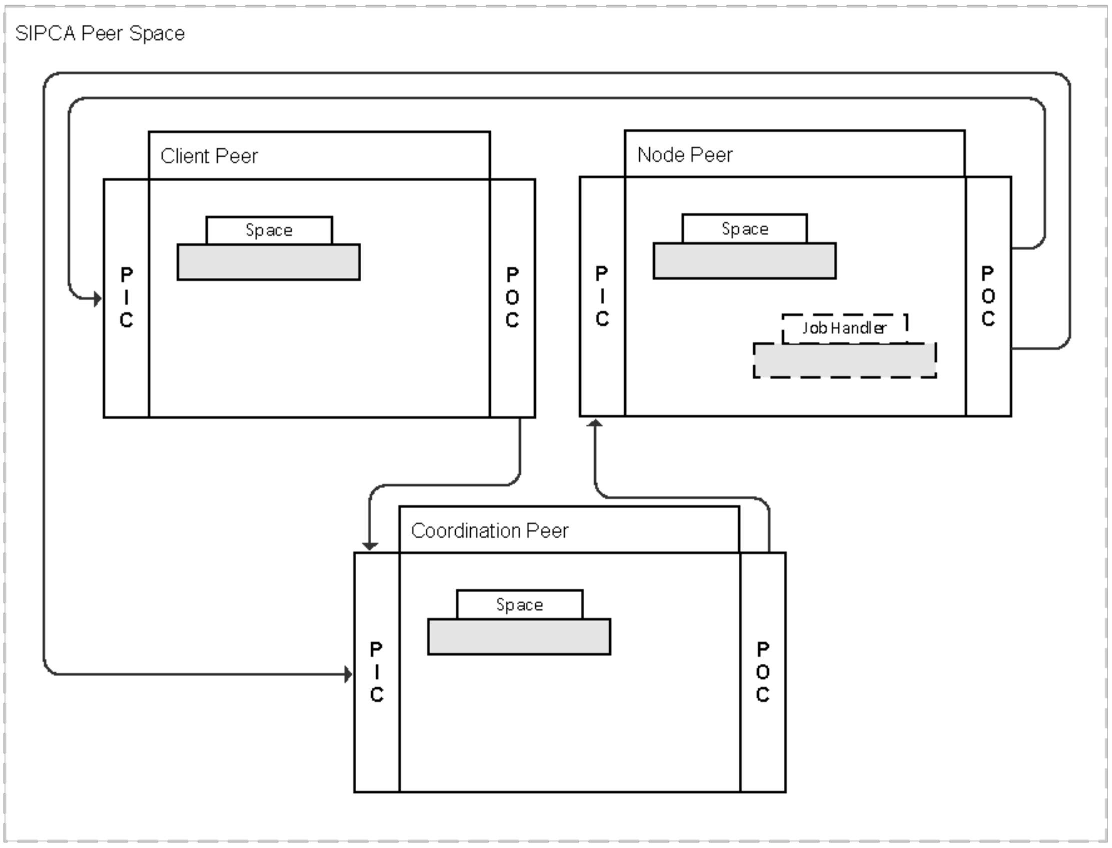

Figure 3.4: Local pattern composition in the framework.

result is sent to the Client Peer, initially sending the request. If a request cannot be fulfilled by a Node Peer, it is forwarded to one or multiple neighboring Node Peers. Thus, a job request is forwarded to only one neighbor, whereas a query request is forwarded to multiple ones. The decision criteria whether a request is fulfillable or not also depends on the Node Peer's type; a Storage Peer just looks up if the required file to satisfy the query is present in its space peer or not, whereas a Worker Peer needs to check if its currently available computational power suffices for the execution of the demanded job.

If a benchmark is run distributed, meaning that multiple SIPCA Peer Space instances contain peers, the number of Coordination Peers in the benchmark equals the number of participating SIPCA Peer Spaces. The role of the Coordination Peers in such a distributed setting, especially the task of clustering but also other responsibilities, are addressed in section 3.3.

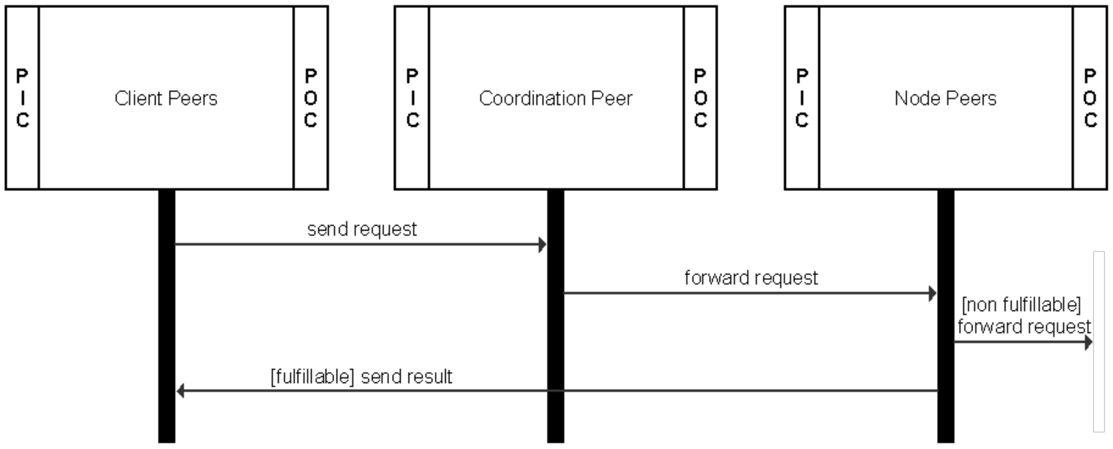

Figure 3.5: Local inter-node collaboration with UML-like sequence diagram.

3.3 Framework Composition

On the top-level the framework consists of two types of Peer Spaces; the SIPCA Peer Space, as discussed in section 3.2, and the Statistics Peer Space, which in fact contains only one Statistics Peer. The collaboration of the framework's components is elaborated in this section.

3.3.1 Synchronization Pattern

In the framework there may exist multiple instances of the SIPCA Peer Space, each possibly running on an own physical machine. Therefore, those Peer Spaces have to be synchronized. This is the responsibility of the Coordination Peer of each SIPCA Peer Space. How they collaborate can be seen in figure 3.6.

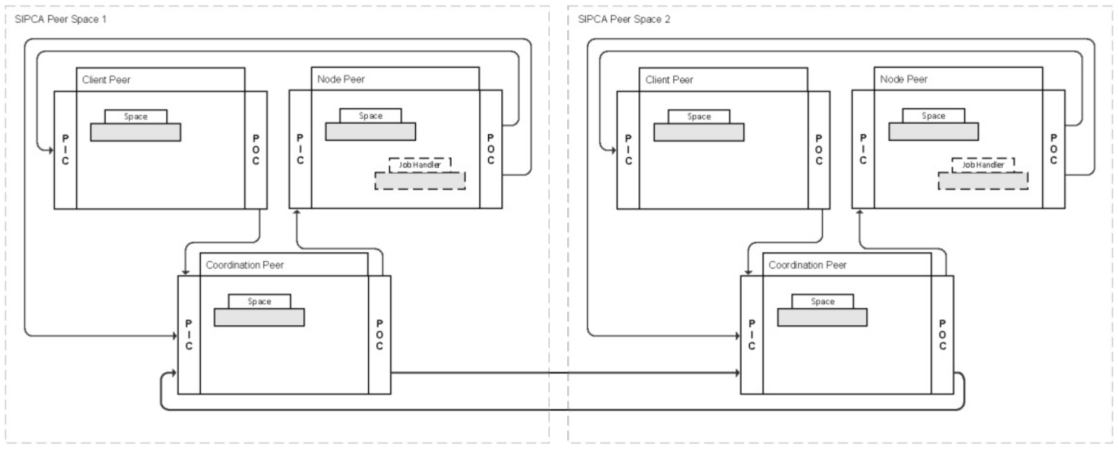

Figure 3.6: Distributed composition of two SIPCA Peer Spaces.

The responsibilities of the Coordination Peer consist of two key objectives: finding the longest-living Coordination Peer in the framework and clustering the Node Peers. The actual clustering is performed by the longest-living Coordination Peer, while the other Coordination Peers provide the longest-living Coordination Peer with information about the Node Peers in their home Peer Space.

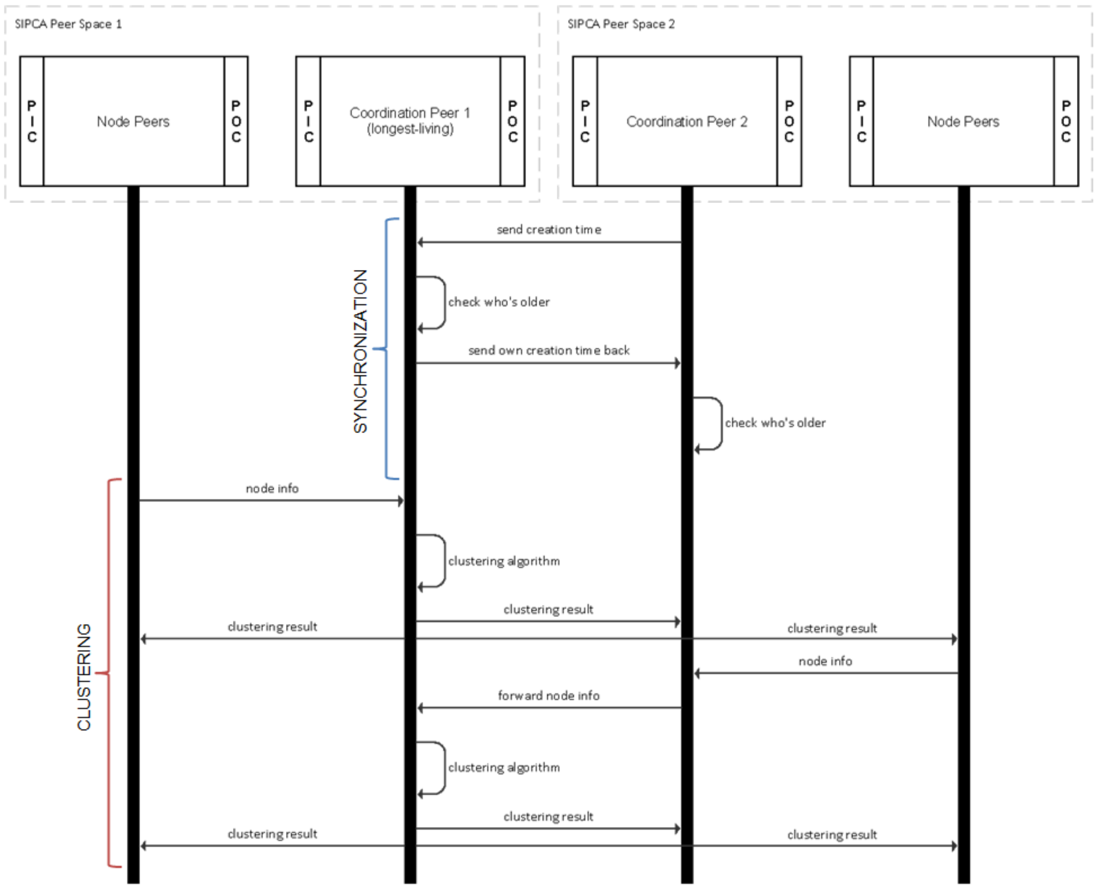

Figure 3.7: Distributed inter-node collaboration on synchronization and clustering.

Figure 3.7 shows the collaboration of peers to keep all instances of SIPCA Peer Spaces synchronized. Therefore, it is essential to know for each Coordination Peer which one of them is the longest-living one. Thus, on start-up a Coordination Peer sends its creation time to all existing Coordination Peers in the framework. They compare it with their own creation time to decide, which of them is older and send their own creation time back to the newly created one. By this means, also the newly created Coordination Peer is able to determine which Coordination Peer is the longest-living one.
Also, when the longest-living Coordination Peer retires, the other Coordination Peers send each other their creation time in order to find the next longest-living. This means, if the longest-living Coordination Peer dies for some reason, the second longest-living one

takes over and so on. This makes the system completely loosely coupled and guarantees that there is no bottleneck regarding throughput.
A Node Peer sends its stored information about itself to the Coordination Peer in its home Peer Space, either on start-up or on changes of the information (e.g. currently available computational power changes on accepting or finishing a job). If the receiving Coordination Peer is the longest-living, it starts clustering. If it is not the longest-living one, the information is forwarded to the longest-living Coordination Peer which then executes the peer clustering algorithm. When clustering is finished, the results are sent to the respective Node Peer.

3.3.1.1 Clustering Trigger Policies

To sum up, there are different ways to trigger the execution of the peer clustering algorithm:

- When the longest-living Coordination Peer receives Node Peer information of a Node Peer of its home SIPCA Peer Space.
- When the longest-living Coordination Peer receives Node Peer information of a Node Peer of a foreign SIPCA Peer Space, forwarded by another Coordination Peer.
- When the longest-living Coordination Peer is informed about a retired SIPCA Peer Space, as the Node Peers of that Space cannot be considered as neighbored nodes any more.

A Node Peer sends its information to its Coordination Peer in following cases:

- on start-up
- on updates of the information, like changing the currently available computational power

Furthermore, the framework is adaptable (cf. section 4.6) regarding the number of Worker Peer information, which shall be available to trigger the execution of the clustering algorithm. The default value for this is 1.

3.3.2 Statistics Pattern

The Statistics Peer Space exists exactly once in the framework and it contains exactly one Statistics Peer. It is essential to carry out the benchmark, as it is required to keep track of the statistical relevant data. Figure 3.8 illustrates the composition of a SIPCA Peer Space and the Statistics Peer Space.

The Statistics Peer receives data from the Client Peer as well as from the Coordination Peer. The data received from the Client Peer contains requests and results, whereas the

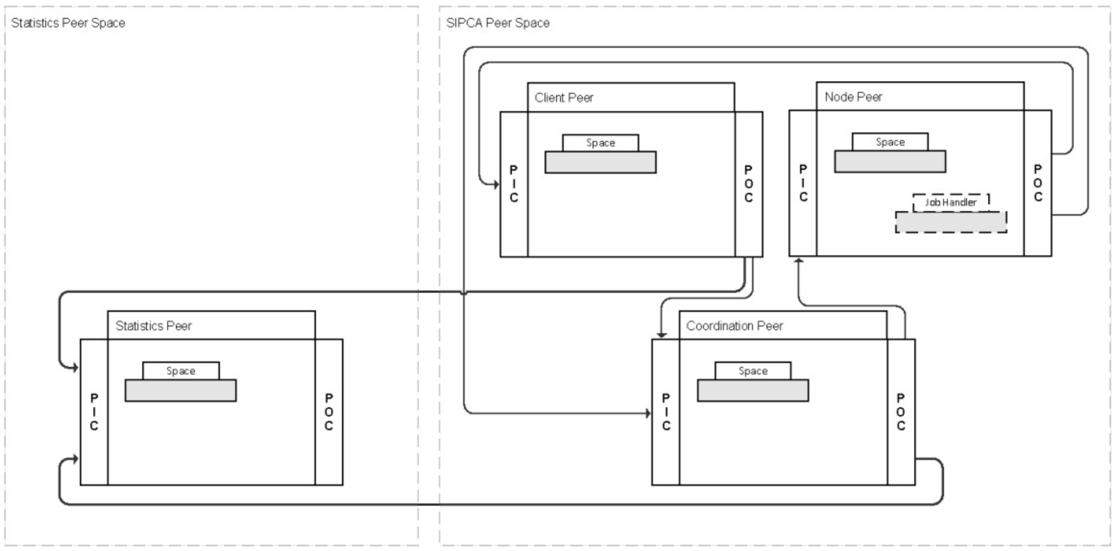

Figure 3.8: Distributed composition of a Statistics Peer Space and a SIPCA Peer Space.

information received from the Coordination Peer contains clustering specific data. The requests and results contain a start and end time, respectively. Thus, the execution time of a request can be calculated.
Consequently, the Statistics Peer uses received data to calculate required information and subsequently documents it. When the Statistics Peer does not receive any data for more than 120 seconds, it is shut down, with the consequence that all SIPCA Peer Spaces also stop. Hence, a benchmark stops when all requests are handled, for which it was possible to be processed. For example, when a file, which is just not available in the network, is queried, processing is not possible.

In figure 3.9 the communication between the peers of a SIPCA Peer Space and the peer contained in the Statistics Peer Space is exemplified. When a Client Peer sends a request to the Coordination Peer, as stated in section 3.2, it also sends a copy to the Statistics Peers. Identically, when the Client Peer receives a result, it transmits a copy to the Statistics Peer.
Furthermore, the longest-living Coordination Peer sends above mentioned information about the peer clustering procedure to the Statistics Peer after the algorithm was executed.

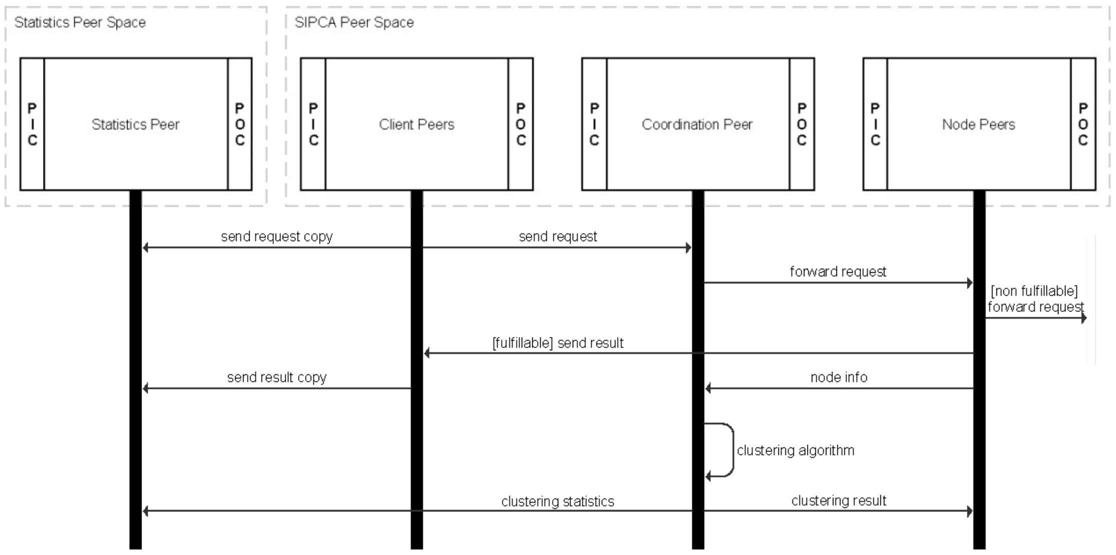

Figure 3.9: Distributed inter-node collaboration on statistics.

3.4 Core Framework Components

The following section describes the components of the proposed framework in detail. The implementation specifics of the framework, including the description of the services and interfaces, are discussed in Chapter 4.
For reasons of clear arrangement the space peer might be placed more than once within a peer in a figure to avoid overlapping arrows.

3.4.1 Entries

The SIPCA framework uses the following types of entries. In brackets, their abbreviations are declared, which are used in descriptions as well as in figures.

- **block clustering entry (blck):** As long as this entry is placed in the space container of a Coordination Peer it is not allowed to execute the clustering algorithm. It is used to prevent clustering before the longest-living Coordination Peer is determined. Thus, every wiring triggering the peer clustering execution is blocked until the Coordination Peers exchanged their creation times. This entry is controlled by a TTL.

- **foreign creation entry (fCrt):** Used by a Coordination Peer to send its creation time to another Coordination Peer.

- **foreign storages entry (fStrgs):** Used by a Coordination Peer to send its stored Storage Peer information to the longest-living Coordination. This information is then merged with the present information in order to start clustering the Storage Peers.

- **foreign workers entry (fWrks):** Used by a Coordination Peer to send its stored Worker Peer information to the longest-living Coordination. This information is then merged with the present information in order to start clustering the Worker Peers.

- **instance retired entry (ret):** Used to inform the Coordination Peers about the fact that a Peer Space retired.

- **job entry (job):** After accepting a job request, the Worker Peers sends this type of entry to its Job Handler Peer.

- **job request entry (jReq):** Embodies the request for a job execution, initially sent from a Client Peer to the Coordination Peer, which then forwards it to a Worker Peer in the according cluster. Then, it is forwarded from one Worker Peer to another until it is accepted. This entry contains, inter alia, the originator, an ID, the minimum required computational power and the job type.

- **job result entry (jRes):** The corresponding answer to a job request. After job execution, the Job Handler Peer sends this entry to the originating Client Peer.

- **job result for update entry (jru):** After finishing a job, the Job Handler Peer sends this entry to the its Worker Peer, in order to trigger updating the released resources, i.e. computational power.

- **oldest coordination peer entry (old):** This type of entry is stored by each Coordination Peer. It contains the space and creation time of the currently longest-living Coordination Peer and whether the holding peer is it or not.

- **parameters entry (par):** On start-up, the Coordination Peer sends this entry to the Statistics Peer. It contains the configuration parameters (cf. section 4.6) of a SIPCA Peer Space, such that they can be documented for clarification of a benchmark.

- **peer info entry (pInfo):** Each peer, excepting the Client Peer, holds such an entry. It serves to store personal information, such as peer name, space name, topic or job interests, etc. The exact content depends on the type of peer storing the information.

- **query request entry (qReq):** Wraps the request for a file query, initially sent from a Client Peer to the Coordination Peer, which then forwards it to a Storage Peer in the according cluster. Then, it is forwarded from one Storage Peer to its neighboring Storage Peers until the required file is found. This entry contains, inter alia, the originator, an ID, the topic and the file name.

- **query result entry (qRes):** The corresponding answer to a query request. After the required file is found, the Storage Peer sends this entry, embodying the requested file, to the originating Client Peer.

- **resource entry (rsrc):** Wraps a file, which can be queried by Client Peers and is stored by Storage Peers.

- **storage clustering result entry (scRes):** Sent by the longest-living Coordination Peer to all other Coordination Peers, Storage Peers and the Statistics Peer to communicate the result of Storage Peer clustering.

- **storage info entry (sInfo):** Used by Storage Peers to send their personal information to the Coordination Peer of their space.

- **storages entry (strgs):** Used by Coordination Peers to store Storage Peer information.

- **supervision entry (sv):** Used by the Statistics Peer to monitor when the last statistical relevant data was received. It is controlled by a TTS.

- **worker clustering result entry (wcRes):** Sent by the longest-living Coordination Peer to all other Coordination Peers, Worker Peers and the Statistics Peer to communicate the result of Worker Peer clustering.

- **worker info entry (wInfo):** Used by Worker Peers to send their personal information to the Coordination Peer of their space.

- **workers entry (wrks):** Used by Coordination Peers to store Worker Peer information.

3.4.2 Coordination Peer

Besides forwarding requests from Client Peers to according Node Peers, the Coordination Peer performs the key functionality of the framework, the execution of the actual peer clustering algorithm. Thus, it has by far the most important role and many sophisticated tasks. To comply with its responsibilities, the Coordination Peer contains eleven wirings, which are shown in figure 3.10, figure 3.11 and figure 3.12 and described in detail in the following.

3.4.2.1 W1: Forward Job Wiring

This wiring simply forwards a job request, received from a Client Peer, to a Worker Peer contained in a cluster, which complies with the request.

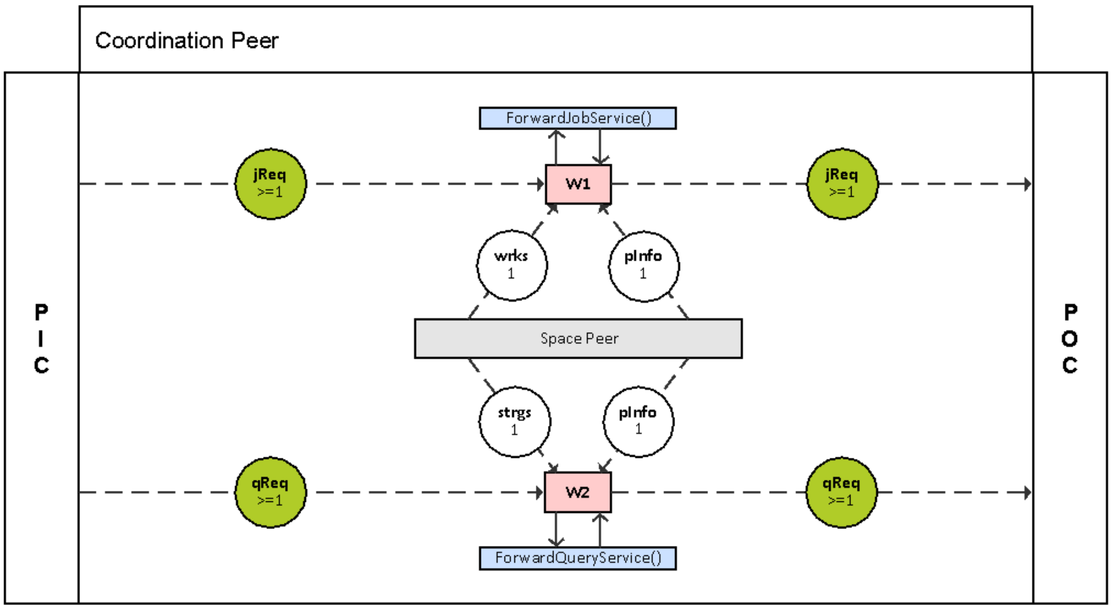

Figure 3.10: Wirings of the Coordination Peer, part 1.

Guards:

1. **jReq (take):** The job request entries to be forwarded to suitable Worker Peers.
2. **wrks (read):** The workers entry contains the list of available Worker Peers.
3. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **jReq:** The dispatched job request entry.

3.4.2.2 W2: Forward Query Wiring

This wiring simply forwards a query request, received from a Client Peer, to a Storage Peer contained in a cluster, which complies with the request.

Guards:

1. **qReq (take):** The query request entries to be forwarded to suitable Storage Peers.
2. **strgs (read):** The storages entry contains the list of available Storage Peers.
3. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **qReq:** The dispatched query request entry.

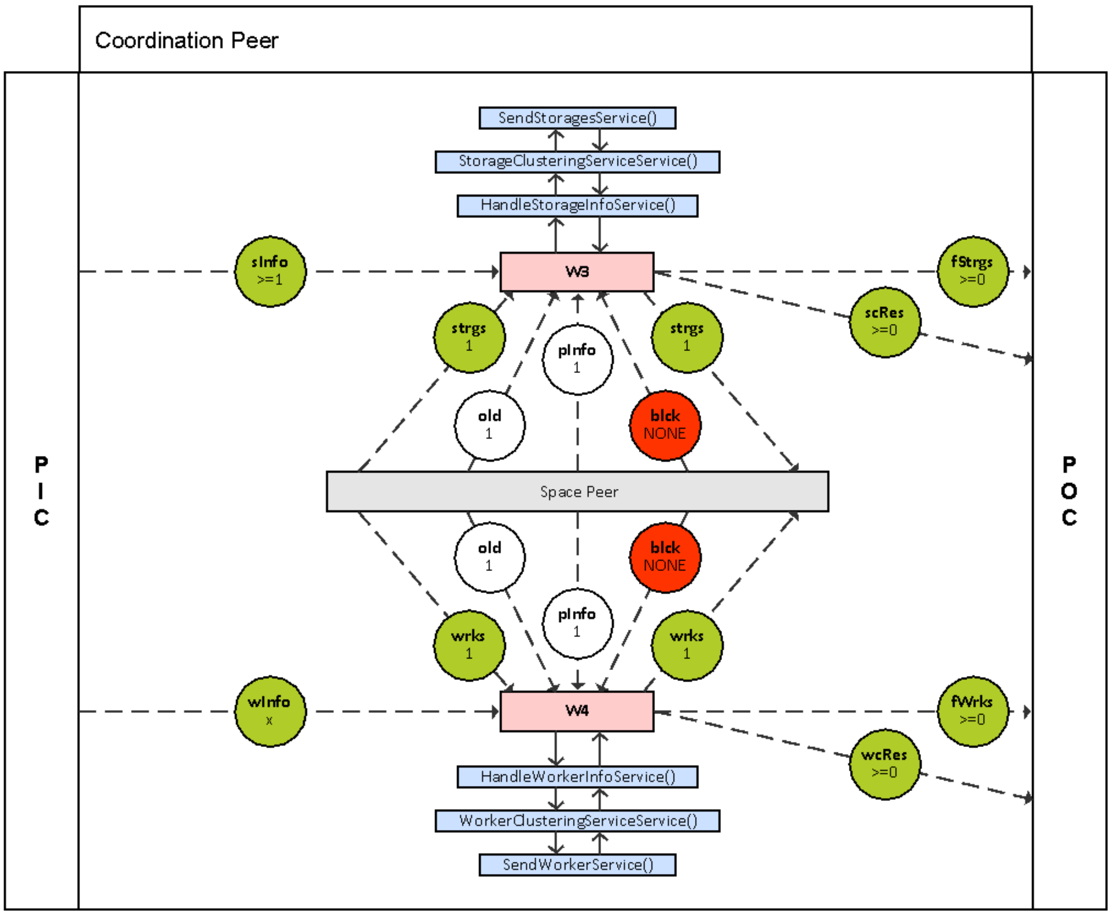

Figure 3.11: Wirings of the Coordination Peer, part 2.

3.4.2.3 W3: Handle Storage Info Wiring

This wiring receives information from a Storage Peer and uses it to update the storages entry. If the Coordination Peer receiving the information is the longest-living, this triggers the execution of the peer clustering algorithm. Otherwise, the storages entry is forwarded to the longest-living Coordination Peer.

Guards:

1. **sInfo (take):** Storage info entries used to update the workers entry.

2. **strgs (take):** Storages entry to be updated.
3. **old (read):** The oldest coordination peer entry provides information about the longest-living Coordination Peer. It is used to determine, whether the current Coordination Peer is the longest-living and shall perform clustering or forward the information to the longest-living Coordination Peer.
4. **pInfo (read):** The peer info entry provides required information.
5. **blck (none):** Coordination Peers have to exchange their creation time first, before they are allowed to trigger clustering.

Actions:

1. **strgs:** The updated storages entry.
2. **fStrgs:** The foreign storages entry forwarded to the longest-living Coordination Peer.
3. **scRes:** The storage clustering result dispatched to the Storage Peers, other Coordination Peers and the Statistics Peer.

3.4.2.4 W4: Handle Worker Info Wiring

This wiring receives information from a Worker Peer and uses it to update the workers entry. The used information is a configurable amount of $x >= 1$ worker info entries. If the Coordination Peer receiving the information is the longest-living, this triggers the execution of the peer clustering algorithm. Otherwise, the workers entry is dispatched to the longest-living Coordination Peer.

Guards:

1. **wInfo (take):** Worker info entries used to update the workers entry.
2. **wrks (take):** Workers entry to be updated.
3. **old (read):** The oldest coordination peer entry provides information about the longest-living Coordination Peer. It is used to determine, whether the current Coordination Peer is the longest-living and shall perform clustering or forward the information to the longest-living Coordination Peer.
4. **pInfo (read):** The peer info entry provides required information.
5. **blck (none):** Coordination Peers have to exchange their creation time first, before they are allowed to trigger clustering.

Actions:

1. **wrks:** The updated workers entry.
2. **fWrks:** The foreign workers entry forwarded to the longest-living Coordination Peer.
3. **wcRes:** The worker clustering result dispatched to the Worker Peers, other Coordination Peers and the Statistics Peer.

3.4.2.5 W5: Merge Foreign Storages Wiring

This wiring receives a foreign storages entry from another Coordination Peer, which is used to merge it with its storages entry and thus, update it. This also triggers the execution of the Storage Peer clustering.

Guards:

1. **fStrgs (take):** Received foreign storages entry to merge with workers entry.
2. **strgs (take):** The storages entry to be updated by merge with the foreign storages entry.
3. **old (read):** The oldest coordination peer entry is used to ensure that the receiving Coordination Peer is the longest-living.
4. **pInfo (read):** The peer info entry provides required information.
5. **blck (none):** Coordination Peers have to exchange their creation time first, before they are allowed to trigger clustering.

Actions:

1. **strgs:** Updated storages entry.
2. **scRes:** The storage clustering result dispatched to the Storage Peers, other Coordination Peers and the Statistics Peer.

3.4.2.6 W6: Merge Foreign Workers Wiring

This wiring receives a foreign workers entry from another Coordination Peer, which is used to merge it with its workers entry and thus, update it. This also triggers the execution of the Worker Peer clustering.

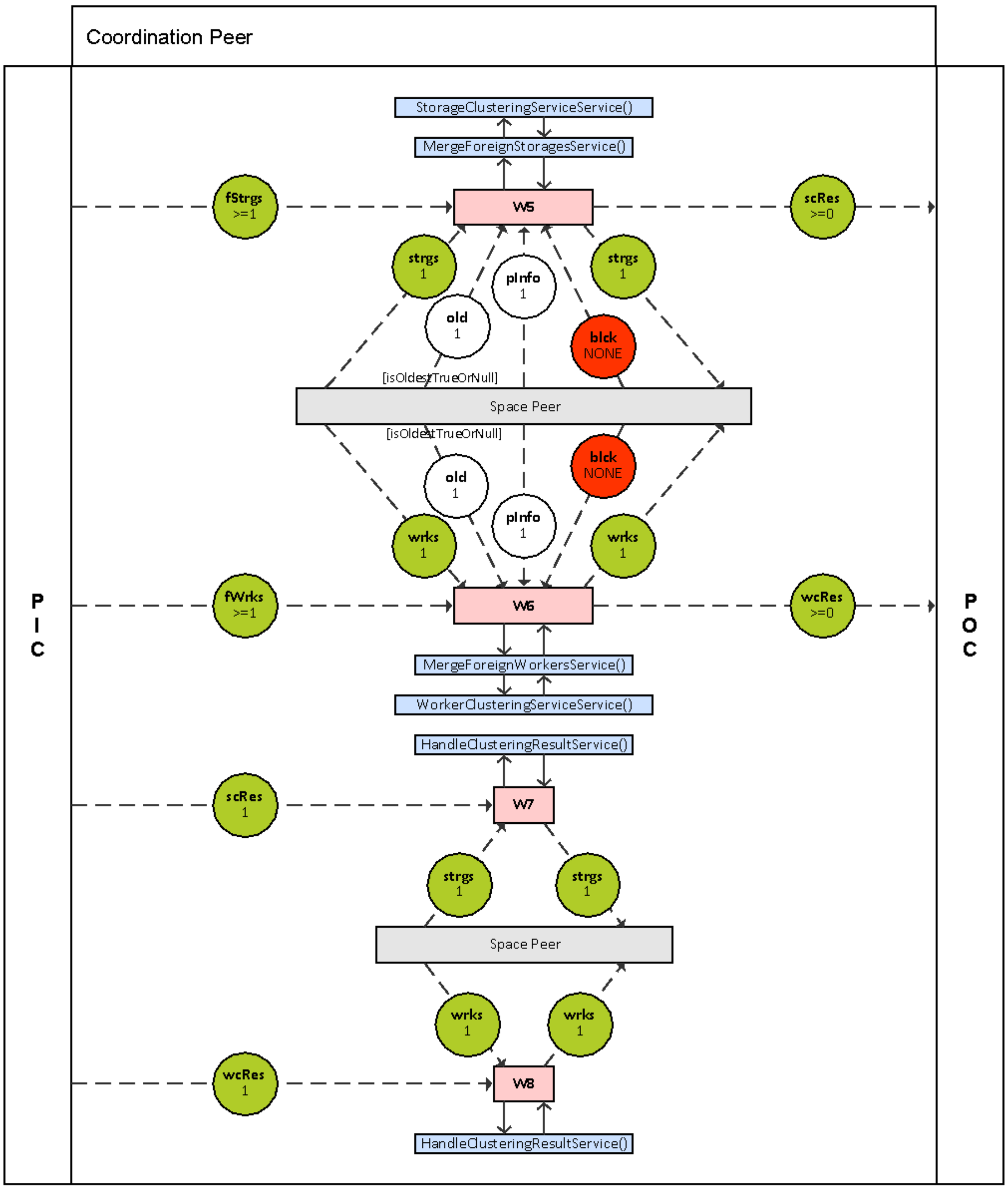
Coordination Peer
StorageClusteringServiceService()
MergeForeignStoragesService()
fStrgs >=1
W5
scRes >=0
strgs 1
pInfo 1
old 1
blck NONE
[isOldestTrueOrNull]
Space Peer
wrks 1
fWrks >=1
W6
wcRes >=0
MergeForeignWorkersService()
WorkerClusteringServiceService()
HandleClusteringResultService()
scRes 1
W7
wcRes 1
W8
PIC
POC

Figure 3.12: Wirings of the Coordination Peer, part 3.

Guards:

1. **fWrks (take):** Received foreign workers entry to merge with workers entry.
2. **wrks (take):** The workers entry to be updated by merge with the foreign workers entry.
3. **old (read):** The oldest coordination peer entry is used to ensure that the receiving Coordination Peer is the longest-living.
4. **pInfo (read):** The peer info entry provides required information.
5. **blck (none):** Coordination Peers have to exchange their creation time first, before they are allowed to trigger clustering.

Actions:

1. **wrks:** Updated workers entry.
2. **wcRes:** The worker clustering result dispatched to the Worker Peers, other Coordination Peers and the Statistics Peer.

3.4.2.7 W7: Handle Storage Clustering Result Wiring

This wiring deals with the Storage Peer clustering result, received from the longest-living Coordination Peer, and uses it to update the stored storages entry.

Guards:

1. **scRes (take):** The received storage clustering result used to update the storages entry.
2. **strgs (take):** Storages entry to be updated.

Actions:

1. **strgs:** The updated storages entry.

3.4.2.8 W8: Handle Worker Clustering Result Wiring

This wiring deals with the Worker Peer clustering result, received from the longest-living Coordination Peer, and uses it to update the stored workers entry.

Guards:

1. **wcRes (take):** The received worker clustering result used to update the workers entry.
2. **wrks (take):** The workers entry to be updated.

Actions:

1. **wrks:** Updated workers entry.

3.4.2.9 W9: Send Creation Time Wiring

This wiring serves to trigger the sending of the own creation time to the Coordination Peers of other Peer Spaces for the first time after start-up.

Guards:

1. **pInfo (take):** Peer info entry used to trigger dispatch.

Actions:

1. **pInfo:** The peer info entry sent to the space container without TTS.
2. **fCrt:** The foreign creation entry dispatched to the other Coordination Peers.

3.4.2.10 W10: Instance Retired Wiring

This wiring is activated as soon as a Peer Space retires. If the longest-living Coordination Peer was part of the retired space, the own creation time is sent to all other Coordination Peers in order to find the next longest-living. Then, if the active Coordination Peer is currently longest-living Coordination Peer, all Node Peers of the retired space are removed from the storages and workers list and the peer clustering algorithm is started.

Guards:

1. **pInfo (take):** The peer info entry provides required information.
2. **ret (take):** The instance retired entry provides information about the retired Peer Space.
3. **old (take):** The oldest coordination peer entry provides information about the longest-living Coordination Peer.

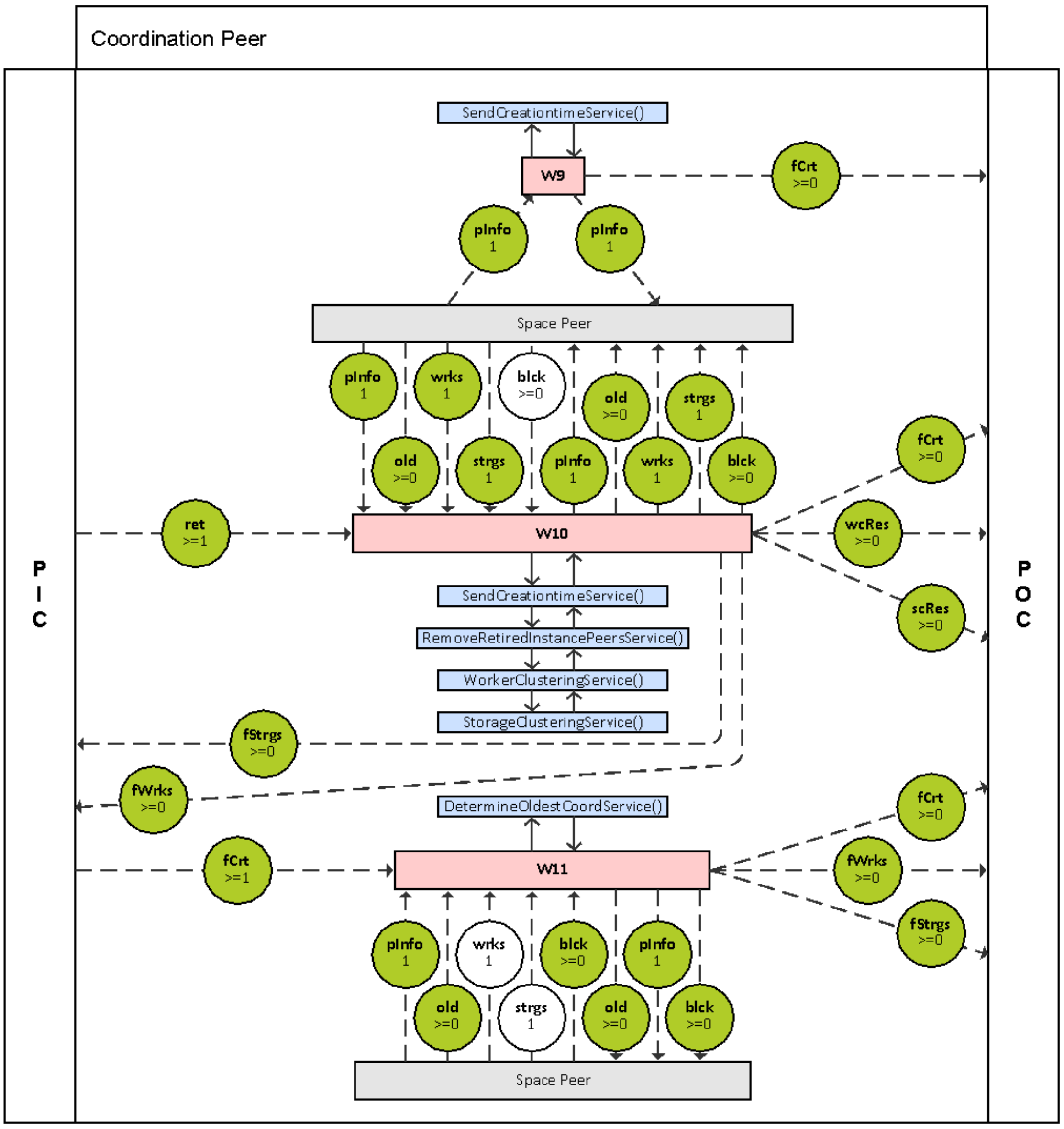

Figure 3.13: Wirings of the Coordination Peer, part 4.

4. **wrks (take):** The workers entry to be updated.

5. **strgs (take):** The storages entry to be updated.

6. **blck (read):** The Coordination Peers have to exchange their creation time first, before they are allowed to trigger clustering.

Actions:

1. **pInfo:** The peer info entry is updated with SIPCA Peer Spaces, which are known at the moment, to guarantee that the creation time is sent only once to a Peer Space.

2. **fCrt:** Foreign creation entry dispatched to the other Coordination Peers.

3. **old:** The updated oldest coordination peer entry.

4. **wrks:** Updated workers entry.

5. **wcRes:** The worker clustering result dispatched to the Worker Peers, other Coordination Peers and the Statistics Peer.

6. **strgs:** Updated storages entry.

7. **scRes:** The storage clustering result dispatched to the Storage Peers, other Coordination Peers and the Statistics Peer.

8. **blck:** If the longest-living Coordination Peer retired, a new block clustering entry is created.

9. **fStrgs:** The updated storages entry is dispatched to the Coordination Peer's own PIC as a foreign storages entry in order to reschedule clustering as soon as the block clustering entry disappears.

10. **fWrks:** The updated storages entry is dispatched to the Coordination Peer's own PIC as a foreign workers entry in order to reschedule clustering as soon as the block clustering entry disappears.

3.4.2.11 W11: Determine Oldest Coordination Peer Wiring

This wiring deals with foreign creation entries, received from other Coordination Peers, in order to determine the new longest-living Coordination Peer. Therefore, creation times are compared. If a new longest-living Coordination Peer is obtained, workers and storages entries are dispatched to it.

Guards:

1. **fCrt (take):** Foreign creation entries received from other Coordination Peers.
2. **pInfo (take):** The peer info entry provides required information.
3. **old (take):** Oldest coordination peer entry to be updated.
4. **wrks (read):** The workers entry may be required to be sent to the new longest-living Coordination Peer.
5. **strgs (read):** The storages entry may be required to be sent to the new longest-living Coordination Peer.
6. **blck (take):** If the current Coordination Peer is also the longest-living, the block clustering entry is removed.

Actions:

1. **old:** Updated oldest coordination peer entry.
2. **pInfo:** The peer info entry is updated with SIPCA Peer Spaces, which are known at the moment, to guarantee that the creation time is sent only once to a Peer Space.
3. **fCrt:** Foreign creation entry dispatched to the originator of the received foreign creation entry.
4. **fWrks:** Foreign workers entry sent to newly obtained longest-living Coordination Peer.
5. **fStrgs:** Foreign storages entry sent to newly obtained longest-living Coordination Peer.
6. **blck:** If the current Coordination Peer is not the longest-living, the block clustering entry is set anew.

3.4.3 Storage Peer

The Storage Peer is a Node Peer in the SIPCA framework. It is supposed to handle file queries and return the corresponding file. Therefore, the Storage Peer entails two wirings, depicted in figure 3.14.

3.4.3.1 W1: Handle Query Wiring

This wiring receives a query either from the Coordination Peer or another Storage Peer. Then, it has to be looked up whether the queried file is available or not. If the file is available, it is dispatched to the originating Client Peer. Otherwise, the query is forwarded to neighboring Storage Peers.

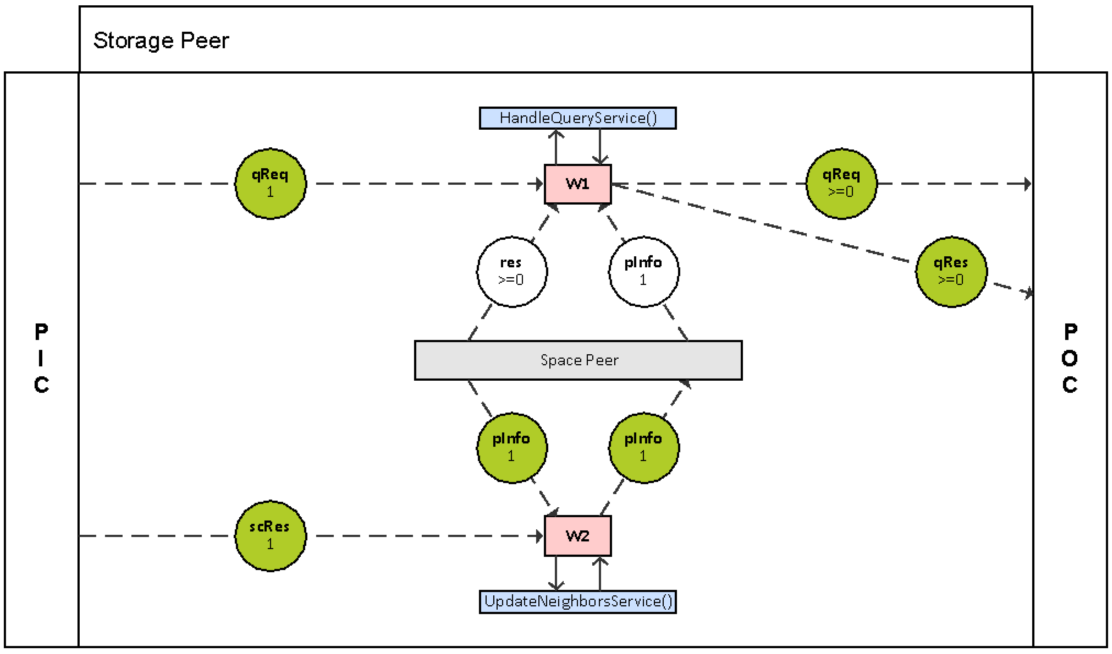

Figure 3.14: Wirings of the Storage Peer.

Guards:

1. **qReq (take):** Query request entry to be handled.
2. **rsrc (read):** The resource entries contain files, may corresponding to the query request.
3. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **qReq:** Forwarded query request entry.
2. **qRes:** The query result entry, dispatched to the querying Client Peer.

3.4.3.2 W2: Update Neighbors Wiring

This wiring uses the received clustering result to update the neighbors of the receiving Storage Peer.

Guards:

1. **scRes (take):** The storage clustering result entry contains the new neighbors of the current Storage Peer.

2. **pInfo (take):** Peer info entry to be updated using the new neighbors.

Actions:

1. **pInfo:** Updated peer info entry with newly obtained neighbors.

3.4.4 Worker Peer

The Worker Peer is a Node Peer in the SIPCA framework. It is supposed to execute jobs, if possible, requested by Client Peers and return the corresponding job result. Whether a job can be executed by a Worker Peer or not, depends on the requested job type and the current available computational power. As can be seen in figure 3.15, the Worker Peer entails three wirings for this, which are explained hereinafter.

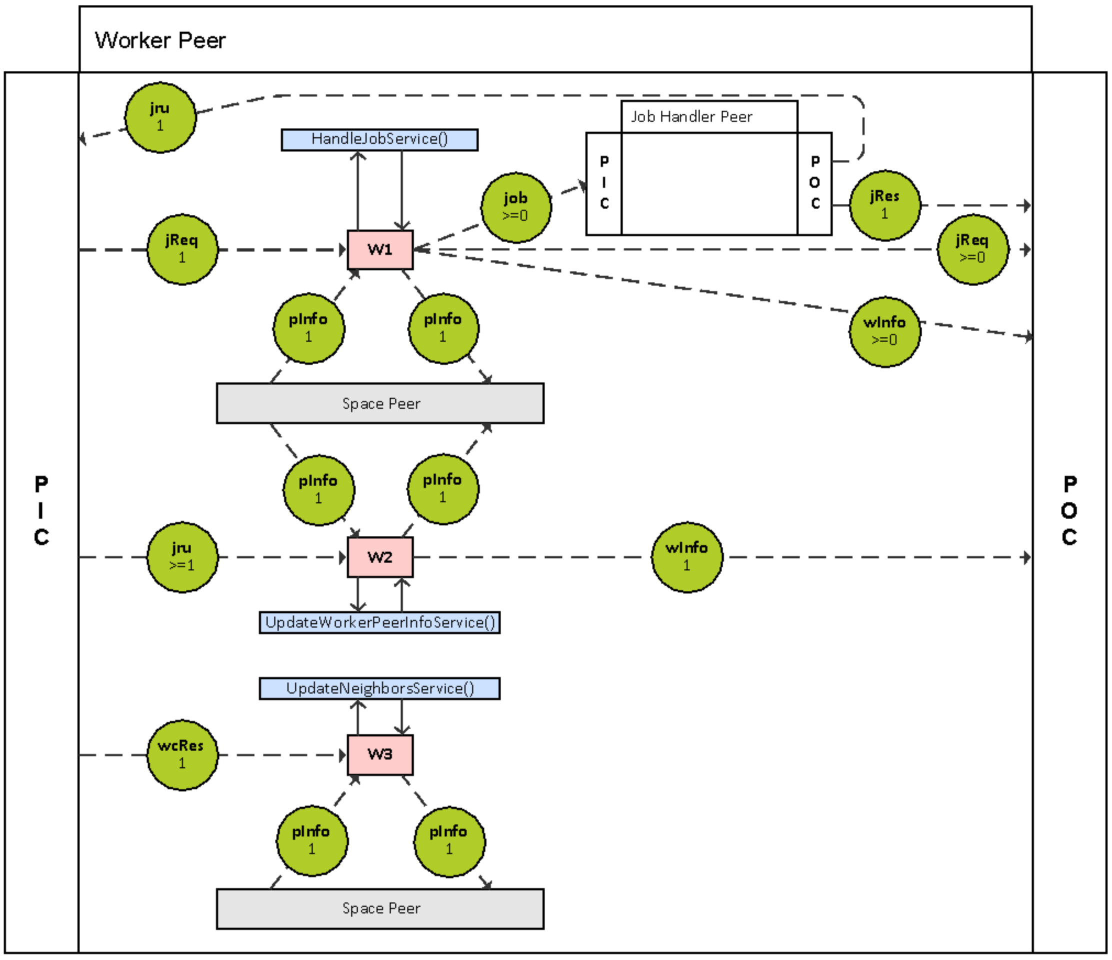

Figure 3.15: Wirings of the Worker Peer.

3.4.4.1 W1: Handle Job Wiring

This wiring receives a job request either from the Coordination Peer or another Worker Peer. Then, it has to be examined whether the job execution is possible or not. If it can be executed, the job is accepted and dispatched to the Worker Peer's Job Handler Peer. Otherwise, the job request is forwarded to a neighboring Worker Peer. When accepting the job, the peer info entry needs to be updated on the allocated resources and sent to the Coordination Peer.

Guards:

1. **jReq (take):** Job request entry to be handled.
2. **pInfo (take):** The peer info entry provides required information.

Actions:

1. **jReq:** Forwarded job request entry.
2. **job:** Job entry dispatched to the Job Handler Peer.
3. **pInfo:** Updated peer info entry with current available computational power.
4. **wInfo:** Worker info entry sent to the Coordination Peer.

3.4.4.2 W2: Update Peer Info Wiring

This wiring simply updates the peer info on released resources and dispatches a copy to the Coordination Peer. For this purpose, it receives information about the completed job from its Job Handler Peer.

Guards:

1. **jru (take):** Job result for update entry used to update the peer info entry.
2. **pInfo (take):** The peer info entry to be updated.

Actions:

1. **pInfo:** Updated peer info entry with current available computational power.
2. **wInfo:** Worker info entry sent to the Coordination Peer.

3.4.4.3 W3: Update Neighbors Wiring

This wiring uses the received clustering result to update the neighbors of the receiving Worker Peer.

Guards:

1. **wcRes (take):** The worker clustering result entry contains the new neighbors of the current Worker Peer.
2. **pInfo (take):** Peer info entry to be updated using the new neighbors.

Actions:

1. **pInfo:** Updated peer info entry with newly obtained neighbors.

3.4.5 Job Handler Peer

The Job Handler Peer is a sub-peer of the Worker Peer. Its task is the actual execution of a requested job. For this purpose, the Job Handler Peer contains one wiring, presented in figure 3.16.

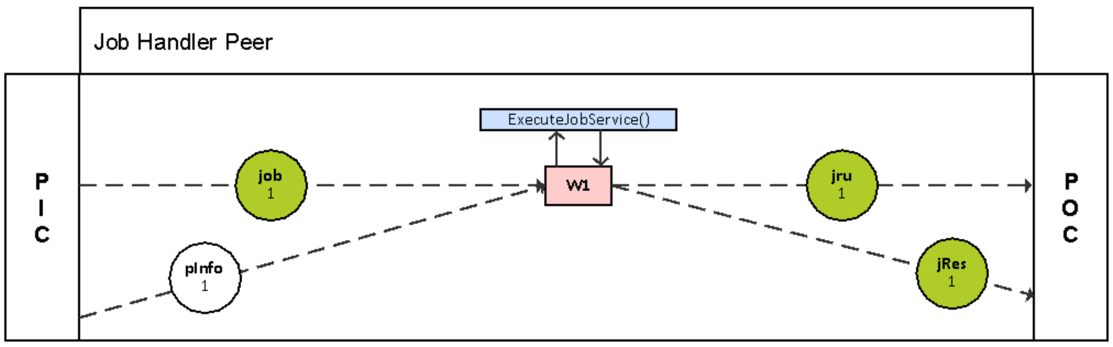

Figure 3.16: Wiring of the Job Handler Peer.

3.4.5.1 W1: Execute Job Wiring

The aim of this wiring is to execute a job, received from its Worker Peer. After finishing, the job result is dispatched to the originating Client Peer and the Worker Peer has to be informed about the completion of the job.

Guards:

1. **job (take):** The job entry to be executed.
2. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **jru:** Job result for update entry to notify the Worker Peer.
2. **jRes:** The job result entry dispatched to the requesting Client Peer.

3.4.6 Client Peer

The Client Peer sends requests to the Coordination Peer of its home Peer Space and eventually receives corresponding results, which are then forwarded to the Statistics Peer. A request can be either a job request or a query request. The Client Peer contains only one wiring, shown in figure 3.17.

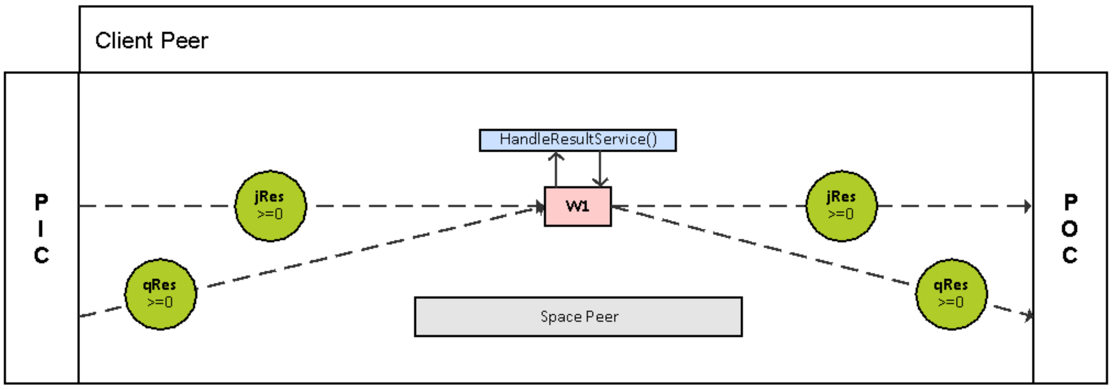

Figure 3.17: Wiring of the Client Peer.

3.4.6.1 W1: Handle Result Wiring

This wiring is activated after receiving a result and simply forwards it to the Statistics Peer.

Guards:

1. **jRes (take):** Job result entry to be forwarded.
2. **qRes (take):** Query result entry to be forwarded.

Actions:

1. **jRes:** Job result entry dispatched to the Statistics Peer.
2. **qRes:** Query result entry dispatched to the Statistics Peer.

3.5 Additional Framework Component

The following peer is not directly part of the core framework. Thus, the SIPCA framework is able to operate smoothly without it, but there won't be any statistically relevant data retained.

3.5.1 Statistics Peer

The Statistics Peer is a peer placed in the Statistics Peer Space. It is meant to receive statistically relevant data and document it. As figure 3.18 illustrates, the Statistics Peer contains six wirings described in detail below.

3.5.1.1 W1: Receive Job Result Wiring

This wiring simply looks up if the corresponding job request for a received job result is available and in this case documents the job result.

Guards:

1. **jReq (take):** The job request entries to be browsed.
2. **jRes (take):** The received job result entry.
3. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **jReq:** The remaining job request are put back in the PIC of the Statistics Peer.
2. **jRes:** The job result entry is put back to the PIC of the Statistics Peer if no corresponding request was found.

3.5.1.2 W2: Receive Query Result Wiring

This wiring looks up if the corresponding query request for a received query result is available and in this case documents the query result.

Guards:

1. **qReq (take):** The query request entries to be browsed.
2. **qRes (take):** The received query result entry.
3. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **qReq:** The remaining query request are put back in the PIC of the Statistics Peer.
2. **qRes:** The query result entry is put back to the PIC of the Statistics Peer if no corresponding request was found.

3.5.1.3 W3: Receive Storage Clustering Result Wiring

This wiring simply documents received storage clustering results received from the longest-living Coordination Peer.

Guards:

1. **scRes (take):** The received storage clustering result entry.
2. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **scRes:** The storage clustering result entry is put back to the PIC of the Statistics Peer if the result file is locked at the moment.

3.5.1.4 W4: Receive Worker Clustering Result Wiring

This wiring simply documents received worker clustering results received from the longest-living Coordination Peer.

Guards:

1. **wcRes (take):** The received worker clustering result entry.
2. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **wcRes:** The worker clustering result entry is put back to the PIC of the Statistics Peer if the result file is locked at the moment.

3.5.1.5 W5: Receive Parameters Wiring

This wiring simply documents received SIPCA parameters received from each Coordination Peer of the benchmark for clarification reasons.

Guards:

1. **par (take):** The received parameters entry.
2. **pInfo (read):** The peer info entry provides required information.

Actions:

1. **par:** The parameters entry is put back to the PIC of the Statistics Peer if the result file is locked at the moment.

3.5.1.6 W6: Check Last Activity Wiring

This wiring is supposed to check when the Statistics Peer was active the last time, i.e when the last data was received.

Guards:

1. **sv (take):** The supervision entry triggers this wiring.
2. **qRes (none):** There must not be a query result entry, which should be recorded in the result file, in the PIC of the Statistics Peer.
3. **jRes (none):** There must not be a job result entry, which should be recorded in the result file, in the PIC of the Statistics Peer.
4. **par (none):** There must not be a parameter entry, which should be recorded in the result file, in the PIC of the Statistics Peer.
5. **scRes (none):** There must not be a storage clustering result entry, which should be recorded in the result file, in the PIC of the Statistics Peer.
6. **wcRes (none):** There must not be a worker clustering result entry, which should be recorded in the result file, in the PIC of the Statistics Peer.

Actions:

1. **sv:** The supervision entry with newly started timer is put back into the PIC of the Statistics Peer.

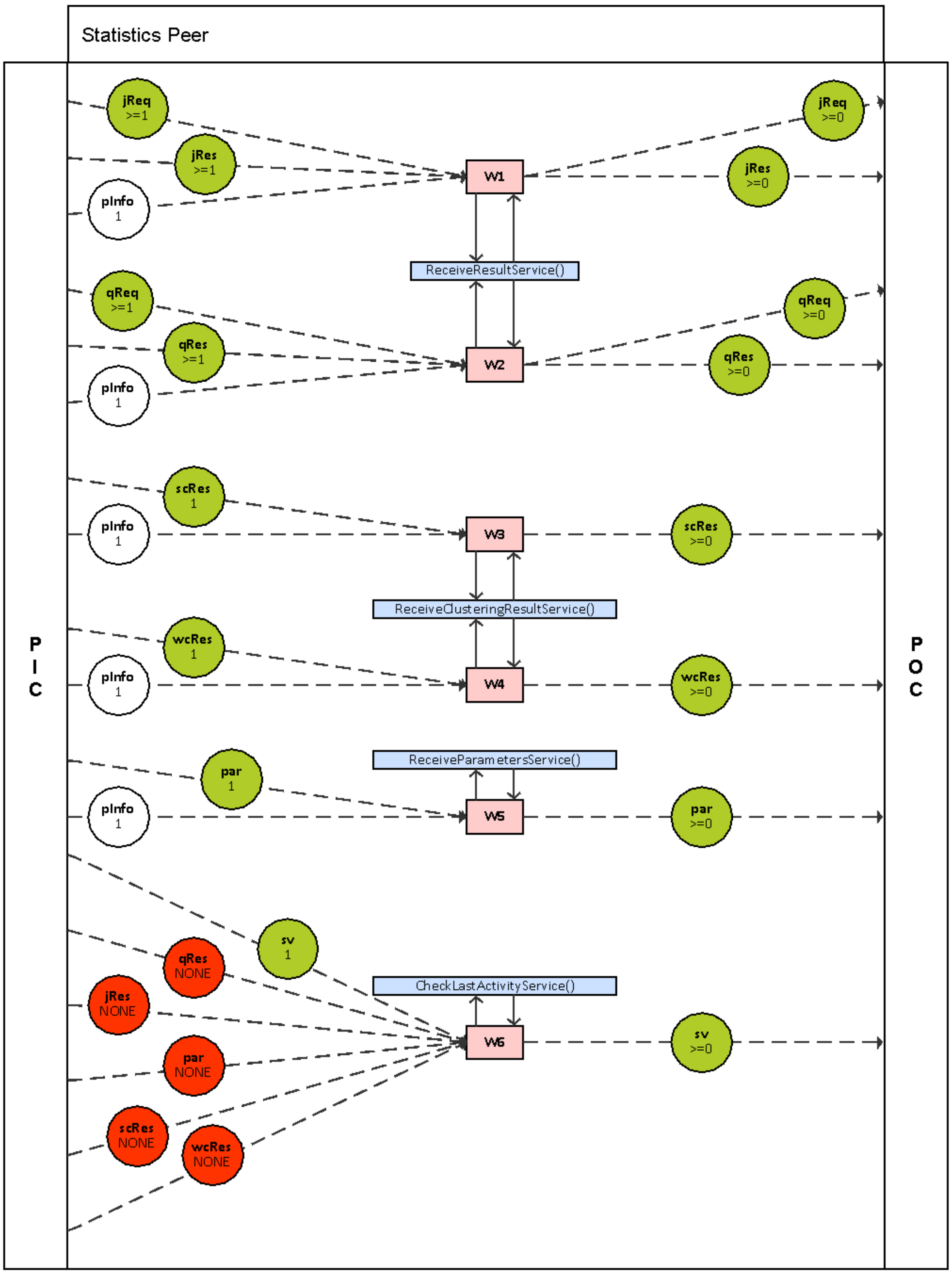

Figure 3.18: Wirings of the Statistics Peer.

3.6 Summary

At the beginning of this Chapter the pattern for a single node in the SIPCA framework is presented. Then, the pattern composition is described. It illustrates how the nodes interact with each other. Afterwards, the framework composition is discussed, which includes the synchronization pattern, the clustering trigger policies and the statistics pattern. Finally, the core components and additional components of the framework are presented in detail, including their wirings and entries, and modelled using the Peer Model notation.

CHAPTER 4

Implementation Details

In this Chapter a more detailed view on the framework implementation is given. The Chapter is structured the following.
Firstly, changes made to the core Peer Model are presented. After a description of the services of the framework, important interfaces and classes of the SIPCA framework are discussed. Furthermore, the metrics which are benchmarked by the framework are introduced. Finally, a description of the framework execution and configuration follows.

4.1 Peer Model Implementation & Extensions

For the implementation of the framework the Peer Model is used, a coordination-based programming model, which was introduced in section 2.2. A detailed description of the Java 8 [49] implementation of the Peer Model is given in [8]. Thus, the SIPCA framework is built on top of this Peer Model implementation, using the same programming language and version.
Nevertheless, in order to enable the framework's full functionality a small extension to the provided Peer Model implementation had to be made. For this extension no additional classes were added to the Peer Model implementation, but only an existing class was extended in their functionality. The added capacity is discussed in detail in the following subsection.

4.1.1 DefaultPeer

At the moment the Peer Model implementation only allows the creation of default peers, i.e. a standard peer having a PIC and a POC. As a consequence, it is not possible to generate a space peer. For this reason additionally to the PIC and POC a new container was introduced - the space container which serves as a space peer for each default peer.

4.2 Services

This section presents a detailed description of the services, whereas the wirings, including their guards and actions, and the framework entry types are described in Chapter 3. The services are numbered corresponding to the numbering of the wirings in section 3.4 and section 3.5, this means for example Service S1 of the Coordination Peer is executed by wiring W1 of the same peer.

4.2.1 Coordination Peer

4.2.1.1 S1: ForwardJobService

This service simply dispatches a received job request to a Worker Peer located in an appropriate cluster without altering it. If no suitable peer is found, the request is put back into the own PIC with a TTS of 5 seconds to handle it later again.

4.2.1.2 S2: ForwardQueryService

This service simply dispatches a received query request to a Storage Peer located in an appropriate cluster without altering it. If no suitable peer is found, the request is put back into the own PIC with a TTS of 5 seconds to handle it later again.

4.2.1.3 S4: HandleWorkerInfoService

This service uses the received worker info entry to update the workers entry. The updated workers entry is then passed either to the WorkerClusteringService or the SendWorkersService, depending on whether the current peer is the longest-living Coordination Peer or not.

4.2.1.4 S4, S6, S10: WorkerClusteringService

This service is executed by the Handle Worker Info Wiring, the Merge Foreign Workers Wiring and the Instance Retired Wiring if the current Coordination Peer is the longest-living. It ensures the actual execution of the clustering algorithm. Then, the workers entry is updated with the result and put back to the space peer. The result itself is wrapped into a worker clustering result entry and dispatched to all available Worker Peers, Coordination Peers and the Statistics Peer.

4.2.1.5 S4: SendWorkersService

This service is executed if the current Coordination Peer is not the longest-living. It is supposed to wrap the updated workers entry into a foreign workers entry and dispatch it to the longest-living Coordination Peer.

4.2.1.6 S3: HandleStorageInfoService

This service uses the received worker info entry to update the storages entry. The updated storages entry is then passed either to the StorageClusteringService or the SendStoragesService, depending on whether the current peer is the longest-living Coordination Peer or not.

4.2.1.7 S3, S5, S10: StorageClusteringService

This service is executed by the Handle Storage Info Wiring, the Merge Foreign Storages Wiring and the Instance Retired Wiring if the current Coordination Peer is the longest-living. It ensures the actual execution of the clustering algorithm. Then, the storages entry is updated with the result and put back to the space peer. The result itself is wrapped into a storage clustering result entry and dispatched to all available Storage Peers, Coordination Peers and the Statistics Peer.

4.2.1.8 S3: SendStoragesService

This service is executed if the current Coordination Peer is not the longest-living. It is supposed to wrap the updated storages entry into a foreign storages entry and dispatch it to the longest-living Coordination Peer.

4.2.1.9 S6: MergeForeignWorkersService

Also this service is only called if the current peer is the longest-living Coordination Peer. It is applied to update the workers entry using the received foreign workers entry. The updated workers entry is then passed to the WorkerClusteringService.

4.2.1.10 S5: MergeForeignStoragesService

This service is as well only called if the current peer is the longest-living Coordination Peer. It is used to update the storages entry with the received foreign storages entry. The updated storages entry is then passed to the StorageClusteringService.

4.2.1.11 S7, S8: HandleClusteringResultService

This service is executed by the Handle Storage Clustering Result Wiring and the Handle Worker Clustering Result Wiring. It uses the received clustering result entry to update the storages and workers entry, respectively. Then, the updated entry is put back to the space peer.

4.2.1.12 S9, S10: SendCreationtimeService

This service is executed by the Send Creation Time Wiring and the Instance Retired Wiring. It is supposed to wrap the creation time of a Coordination Peer into a foreign creation entry and dispatch it to all available Coordination Peers. Furthermore, if this service is triggered after a Peer Space retired, the list of known instances is updated.

4.2.1.13 S10: RemoveRetiredInstancePeersService

This service is executed only if the current Coordination Peer is the longest-living. It is applied to remove all Storage Peers of the storages entry and all Worker Peers of the workers entry, which were contained in the retired Peer Space, as they are not available as neighbors anymore. Then, the updated storages entry and workers entry is passed to the StorageClusteringService and WorkerClusteringService, respectively. If clustering is blocked at the moment by a block clustering entry placed in the PIC, the storages and workers are wrapped into a foreign storages entry and foreign workers entry, respectively, which are put in the own PIC to trigger clustering as soon as it is enabled.

4.2.1.14 S11: DetermineOldestCoordService

This service is used to find the longest-living Coordination Peer in the SIPCA framework. Therefore, received foreign creation entries are compared to the own creation time. The originator is stored in the known instances list and the own creation time is dispatched to it, wrapped into a foreign creation entry. Moreover, the longest-living Coordination Peer is stored in the space peer as oldest coordination peer entry. If a new longest-living Coordination Peer was obtained, the workers and storages entries are dispatched to it as foreign storages entry and foreign workers entry.
If the current peer is the longest-living Coordination Peer, the block clustering entry is not put back to enable clustering immediately. Otherwise, the taken block clustering entry is put back into the own PIC.

4.2.2 Node Peer

4.2.2.1 S2, S3: UpdateNeighborsService

This service is executed by the Update Neighbors Wiring of the Storage Peer and the Worker Peer. It is applied to update the neighboring peers in the peer info entry using the received clustering result.

4.2.3 Storage Peer

4.2.3.1 S1: HandleQueryService

This service browses the Storage Peer's stored files for the file demanded in the received query request. If the file is found, it is wrapped in a query result entry and dispatched to the originating Client Peer. Otherwise, the request is forwarded to all neighboring Storage Peers in the same cluster, adding the information that the current Storage Peer does not have the demanded file such that the request is not sent back to it. If no such peer is available, the request is put back into the own PIC with a TTS of 5 seconds to handle it later again.

4.2.4 Worker Peer

4.2.4.1 S1: HandleJobService

This service is supposed to check if a received job request can be handled by the current Worker Peer or not. If the job type matches with the job type interests of the peer and the minimum required computational power of the job is smaller or equal to the currently available computational power of the peer, the job is accepted and put into the PIC of its Job Handler Peer. Additionally, the peer info is updated with the newly occupied resources and put into the space peer. This info is also wrapped into a worker info entry and dispatched to the Coordination Peer. If the job is not accepted, the request is forwarded to a Worker Peer within the same cluster. Also here the information is added that the current peer is unable to fulfill the request. But when all neighbors of a peer are on the list of unable peers, the list is cleared. This means those peers can be addressed again, as there is the possibility that they have more free resources now. If no suitable neighbor was found, the request is put back into the own PIC with a TTS of 5 seconds to handle it later again.

4.2.4.2 S2: UpdateWorkerPeerInfoService

After finishing a job, this service is used to update the peer info with the newly released resources. Just as in HandleJobService this information is sent to the Coordination Peer.

4.2.5 Job Handler Peer

4.2.5.1 S1: ExecuteJobService

This service receives a job and waits 3 seconds. Then, the job result entry, corresponding to the job request, is created and dispatched to the originating Client Peer. Furthermore, a job result for update entry is placed in the PIC of the Job Handler Peer's Worker Peer to trigger the update of the Worker Peer's peer info entry with the released resources.

4.2.6 Client Peer

4.2.6.1 S1: HandleResultService

This service adds an end time to a received job or query result, corresponding to a request this Client Peer originated, and dispatches it to the Statistics Peer.

4.2.7 Statistics Peer

4.2.7.1 S1, S2: ReceiveResultService

This service is executed by the Receive Job Result Wiring and the Receive Query Result Wiring. Thus, it receives either a query or job result and looks for the corresponding request.

4.2.7.2 S3, S4: ReceiveClusteringResultService

Similar to the ReceiveResultService, this service receives either a storage or worker clustering result. It calculates some values as required and documents them in a file.

4.2.7.3 S5: ReceiveParametersService

This service simply uses the received parameters of a newly joined SIPCA Peer Space and documents them in a file to clarify the benchmarks.

4.2.7.4 S6: CheckLastActivityService

This service is executed every 2 minutes to assure the result file was modified during the last two minutes. If this is not the case, the benchmark is stopped. Otherwise, the supervision entry is simply put in the Statistics Peer's PIC again with a TTS of 2 minutes.

4.3 Interfaces and Classes

In this section the interface provided for the implementation of the clustering algorithms is presented, as well as classes created to store the Node Peers and classes generally supporting the functionality of the framework.

The `NodePeer` class, shown in figure 4.1, embodies NodePeers and includes all properties Storage Peers and Worker Peers have in common. Additionally, it provides methods useful for both of them, e.g. with `isNewer(n)` it can be determined, which of two Node Peers was created later and thus, represents the current status of a peer.
The classes `StoragePeer` and `WorkerPeer` are used to represent Storage Peers and Worker Peers, respectively. Hence, they have attributes in compliance with the respective type of peer.

In order to have the same conditions for benchmarking the different algorithms, it is necessary to have exactly the same entries in a SIPCA Peer Space. Therefore, when a benchmark is started with a random initialization, which can be configured on execution (cf. section 4.6), the created entries are serialized. Thus, they can be retrieved in the next benchmark, started with non-random initialization. Figure 4.2 portrays the class `EntrySerialization`, serving for this purpose. In fact, this only works if the SIPCA Peer Space instance is started with the same amount of Worker, Storage and Client Peers.

The `IAlgorithm` interface, depicted in figure 4.3, represents the API for the clustering algorithms. This allows the decision of the respective implementation during runtime.
In order to as well allow to choose between different types of distance measurement (cf. section 5.2.1) and evaluation methods (cf. section 4.4), the `IDistance` and `IEvaluation` interfaces are implemented. Thus, they can be used and extended easily. Both interfaces are illustrated in figure 4.4 and figure 4.5.

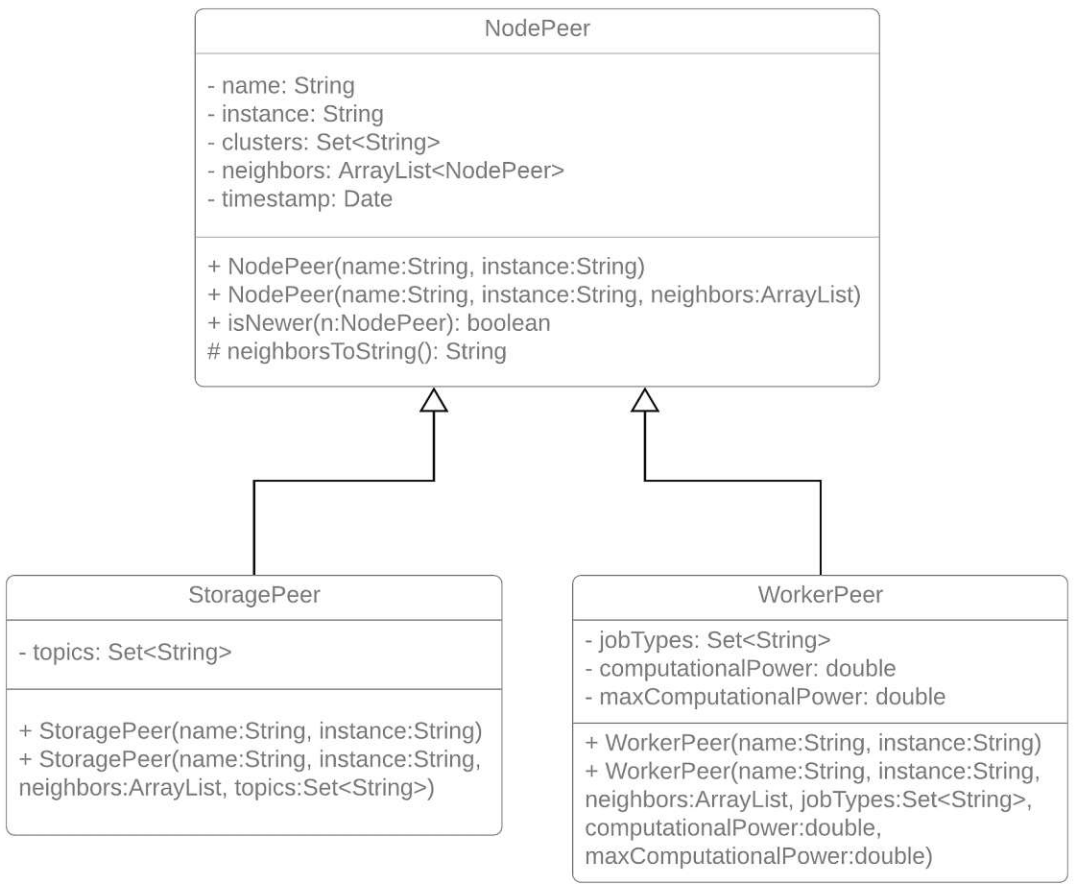
NodePeer
- name: String
- instance: String
- clusters: Set<String>
- neighbors: ArrayList<NodePeer>
- timestamp: Date
+ NodePeer(name:String, instance:String)
+ NodePeer(name:String, instance:String, neighbors:ArrayList)
+ isNewer(n:NodePeer): boolean
# neighborsToString(): String
StoragePeer
- topics: Set<String>
+ StoragePeer(name:String, instance:String)
+ StoragePeer(name:String, instance:String, neighbors:ArrayList, topics:Set<String>)
WorkerPeer
- jobTypes: Set<String>
- computationalPower: double
- maxComputationalPower: double
+ WorkerPeer(name:String, instance:String)
+ WorkerPeer(name:String, instance:String, neighbors:ArrayList, jobTypes:Set<String>, computationalPower:double, maxComputationalPower:double)

Figure 4.1: Class NodePeer with its sub-classes StoragePeer and WorkerPeer. For reasons of simplicity, getter and setter methods are omitted.

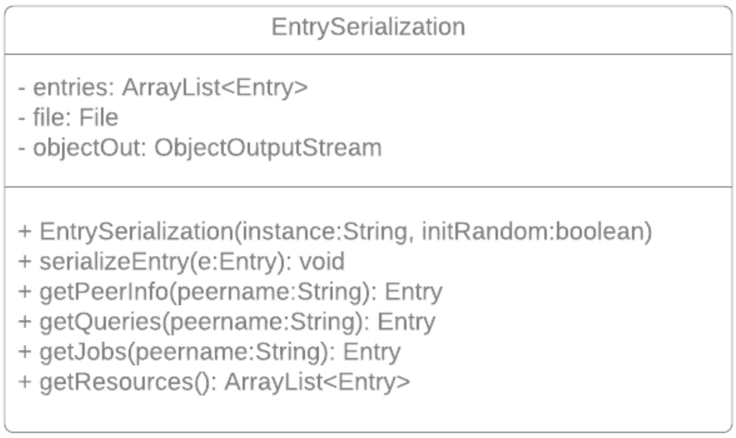
EntrySerialization
- entries: ArrayList<Entry>
- file: File
- objectOut: ObjectOutputStream
+ EntrySerialization(instance:String, initRandom:boolean)
+ serializeEntry(e:Entry): void
+ getPeerInfo(peername:String): Entry
+ getQueries(peername:String): Entry
+ getJobs(peername:String): Entry
+ getResources(): ArrayList<Entry>

Figure 4.2: Class EntrySerialization. For reasons of simplicity, getter and setter methods are omitted.

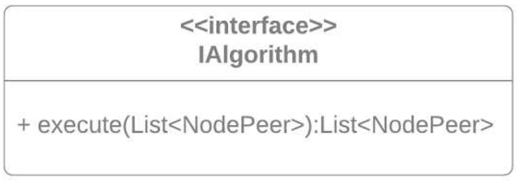

Figure 4.3: Interface IAlgorithm.

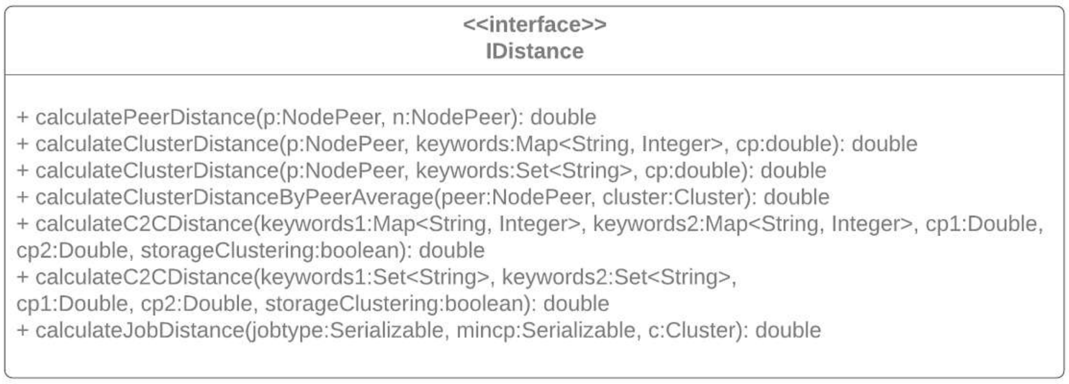
<<interface>>
IDistance
+ calculatePeerDistance(p:NodePeer, n:NodePeer): double
+ calculateClusterDistance(p:NodePeer, keywords:Map<String, Integer>, cp:double): double
+ calculateClusterDistance(p:NodePeer, keywords:Set<String>, cp:double): double
+ calculateClusterDistanceByPeerAverage(peer:NodePeer, cluster:Cluster): double
+ calculateC2CDistance(keywords1:Map<String, Integer>, keywords2:Map<String, Integer>, cp1:Double, cp2:Double, storageClustering:boolean): double
+ calculateC2CDistance(keywords1:Set<String>, keywords2:Set<String>, cp1:Double, cp2:Double, storageClustering:boolean): double
+ calculateJobDistance(jobtype:Serializable, mincp:Serializable, c:Cluster): double

Figure 4.4: Interface IDistance.

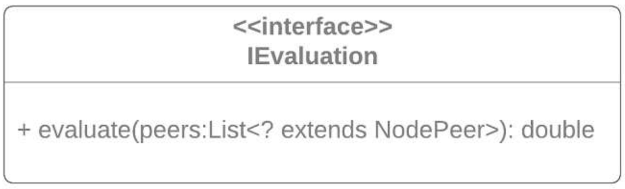

Figure 4.5: Interface IEvaluation.

4.4 Metrics

As the primary task of the framework is to benchmark peer clustering algorithms, metrics need to be defined in order to allow the evaluation of these algorithms. Therefore, in this section the implemented metrics are described and an overview of the output provided by the framework is given.

4.4.1 Benchmark Output

The outputs, which are provided by the framework after a benchmark has ended, are described in table 4.1. On the one hand it provides information about the general setup of the execution and on the other hand, of course, it offers the metrics to evaluate the benchmark.

4.4.2 Davies-Bouldin index

The Davies-Bouldin index [11] shall indicate the similarity of clusters. Therefore, it uses the following formula:

$$DBI = \frac{1}{n}\sum_{i=1}^{n} max_{i\neq j}(\frac{S_i + S_j}{d(c_i, c_j)}) \tag{4.1}$$

where n is the number of clusters, S_i is the average distance between the centroid of cluster i and all peers within cluster i, and $d(c_i, c_j)$ is the distance between the centroid of cluster i and cluster j.
The Davies-Bouldin index is non-negative. The smaller the index, the better is the clustering result.

4.4.3 Dunn index

The Dunn index [14, 15] is a metric aiming at minimizing the the intra-cluster distance, while having well-separated clusters (i.e. maximizing the inter-cluster distance). It uses the following formula:

$$DI = \frac{min_{1\leq i<j\leq n}(d(i,j))}{max_{1\leq k\leq n}(d'(x_k, y_k))} \tag{4.2}$$

where $d(i,j)$ is the distance between the clusters i and j, and $d'(x_k, y_k)$ is the distance of any pair of peers within cluster k.
The higher the Dunn index is, the better is the result of the peer clustering.

4.4.4 Silhouette coefficient

The Silhouette coefficient [39, 23] shall evaluate the validity of a clustering result and is often used to find the optimal number of clusters. It is calculated as follows:

$$a(i) = \frac{\sum d(i,j)}{|C_i| - 1} \tag{4.3}$$

which is the average distance of peer i to all other peers within the same cluster C_i.

$$b(i) = min_{C \neq C_i}(d(i,C)) \tag{4.4}$$

where $d(i,C)$ is the average distance between peer i to all peers of cluster C, for all clusters $C \neq C_i$. Then, those formulas are used to calculate the Silhouette of i $s(i)$:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{4.5}$$

The overall average Silhouette width, here called Silhouette coefficient, $s(k)$ is used to measure the validity of the clustering result:

$$s(k) = \frac{\sum_{i=1}^{n} s(i)}{n} \tag{4.6}$$

where n is the number of peers which were clustered.
The Silhouette coefficient is ranged between -1 and 1. The higher the coefficient, the better is the peer clustering result.

4.4.5 Averaged Dissimilarity coefficient

The Averaged Dissimilarity coefficient is a simple metric to evaluate a peer clustering result and was developed in the course of this thesis. The calculation works as follows:

First, the average distance of a peer i to all other peers in the same cluster C_x is calculated:

$$a(i, C_x) = \frac{\sum_{i=1}^{|C_x|} d(i,j)}{|C_x| - 1} \tag{4.7}$$

This is done for every peer in the cluster C_x. Then, the average cluster within distance for cluster C_x is calculated by summing up all object distances and dividing them through the number of cluster members:

$$b(C_x) = \frac{\sum_{i=1}^{|C_x|} a(i, C_x)}{|C_x|} \tag{4.8}$$

After that, the Averaged Dissimilarity coefficient is calculated by:

$$ADC = \frac{\sum_{x=1}^{k} b(C_x)}{k} \tag{4.9}$$

where k is the number of clusters.
Obviously, the smaller the Averaged Dissimilarity coefficient is, the better is the clustering result.

Metric	Description
clustering algorithm	The clustering algorithm which was used for the benchmark.
number of Worker Peers	The overall amount of Worker Peers participating in the benchmark.
number of Storage Peers	The overall amount of Storage Peers participating in the benchmark.
number of Client Peers	The overall amount of Client Peers participating in the benchmark.
number of query requests	The overall amount of query requests sent in the benchmark.
number of job requests	The overall amount of job requests sent in the benchmark.
number of worker infos	The number of worker info entries which are required to trigger the execution of the clustering algorithm.
distance	Type of distance measurement.
algorithm parameters	The parameter settings of the executed peer clustering algorithm.
number of algorithm executions (Worker Peers)	Number of algorithm executions for Worker Peer clustering during one benchmark.
number of algorithm executions (Storage Peers)	Number of algorithm executions for Storage Peer clustering during one benchmark.
execution time (Worker Peers)	Accumulated, averaged execution time of Worker Peer clustering.
execution time (Storage Peers)	Accumulated, averaged execution time of Storage Peer clustering.
Davies-Bouldin index	Shows the averaged Davies-Bouldin index of the clustering result. Based on [11] and shortly described in section 4.4.2.
Dunn index	Shows the averaged Dunn index of the clustering result. Based on [14] and [15] and shortly described in section 4.4.3.
Silhouette coefficient	Shows the averaged Silhouette coefficient of the clustering result. Based on [39] and [23], and shortly described in section 4.4.4.
Averaged Dissimilarity coefficient	Shows the Averaged Dissimilarity coefficient of the clustering result, described in detail in section 4.4.5.

Table 4.1: Benchmark metrics.

4.5 Framework Execution

The framework is provided in two executable jar-files:

- `peermodel-sipca-framework-statistics.jar`
- `peermodel-sipca-framework.jar`

The first one is used to run a new Statistics Peer Space instance, whereas the latter starts a new SIPCA Peer Space instance. The parameters which should be provided to the `peermodel-sipca-framework.jar` executable are discussed in detail in Section 4.6.

The `peermodel-sipca-framework-statistics.jar` must not be executed more than once per benchmark. By contrast, the `peermodel-sipca-framework.jar` file is not restricted regarding the amount of executions, due to the fact that a benchmark may involves multiple SIPCA Peer Spaces.
Although the execution order of the jar-files is not strictly specified, it is advisable to start the Statistics Peer Space first, since the SIPCA Peer Spaces dispatch entries to the Statistics Peer from the beginning of execution.

4.6 Framework Configuration

The SIPCA framework is configured by adding parameters directly or by adding a configuration file containing the parameters when executing the `peermodel-sipca-framework.jar` file. These parameters are described in table 4.2 and table 4.3. To allow a quick start-up of only one SIPCA Peer Space instance many parameters have a default value.

Usage	Parameter	Description
Position 0	instance name	Defines the unique name of the SIPCA Peer Space instance. With this name the instance can be identified and addressed in the network. **Default:** -
noWorker	number of Worker Peers	Defines the number of Worker Peers hosted by the created SIPCA Peer Space instance. **Default:** 5
noStorage	number of Storage Peers	Defines the number of Storage Peers hosted by the created SIPCA Peer Space instance. **Default:** 5

Table 4.2: SIPCA Peer Space parameters, part 1.

To start a SIPCA Peer Space instance, for example, the command stated in listing 4.1 can be used.

Usage	Parameter	Description
noClient	number of Client Peers	Defines the number of Client Peers hosted by the created SIPCA Peer Space instance. **Default:** 5
noJobs	number of jobs	Defines the amount of job requests created by each Client Peer. **Default:** 1
noQueries	number of queries	Defines the amount of query requests created by each Client Peer. **Default:** 1
clusteringAlgorithm	clustering algorithm	Defines the clustering algorithm which shall be used. For a more detailed explanation see section 4.6.1. **Default:** KMeans
initRandom	random initialization	Defines whether the peers shall be randomly initialized or not. For a more detailed explanation see section 4.6.2. **Default:** true
resourcePath	resource path	Defines the path to the required resource folder. For a more detailed explanation see section 4.6.3. **Default:** -
noWorkerInfos	number of worker infos	The number of worker info entries which are required to trigger the execution of the clustering algorithm. **Default:** 1
reclusterStorages	recluster Storage Peers	Defines whether the Storage Peers shall be reclustered or not. For a more detailed explanation see section 4.6.4. **Default:** false
distance	type of distance measurement	Defines which type of distance measurement shall be used. The choice is between Euclidean and Jaccard distance measurement. For a more detailed explanation see section 5.2.1. **Default:** Euclidean

Table 4.3: SIPCA Peer Space parameters, part 2.

```
java -jar peermodel-sipca-framework.jar test50 noWorker=50
noStorage=50 noClient=50 noJobs=3 noQueries=3
clusteringAlgorithm=KMeans initRandom=false
resourcePath=examplePath/testdata/50 noWorkerInfos=1
reclusterStorages=true noClusters=10
```

Listing 4.1: Command to start a SIPCA Peer Space instance.

Due to the fact that each SIPCA Peer Space instance possibly can execute different clustering algorithms with a different number of worker peer entries triggering this execution, it is possible to mix clustering algorithms up during the benchmark. Nevertheless, the benchmarks of this work will not combine different algorithms and thus, use the same algorithm and amount of required worker infos for each created SIPCA Peer Space.

4.6.1 Clustering Algorithm

The following encodings shall be used for algorithm parameterization:

- ABC: Artificial Bee Colony
- ABCK: Artificial Bee Colony combined with K-Means
- AntClustering: Ant-based Clustering
- AntKmeans: Ant K-Means
- FuzzyCMeans: Fuzzy C-Means
- GeneticKMeans: Genetic K-Means
- Hierarchical: Hierarchical Clustering
- KMeans: K-Means
- PSO: Particle Swarm Optimization
- SlimeMold: Slime Mold
- SlimeMoldK: Slime Mold K-Means

4.6.2 Random Initialization

If peers are initialized randomly, this includes the following. A Storage Peer receives one to three topic interests, randomly chosen from the available topics of this SIPCA Peer Space instance stored in the resource folder (cf. section 4.6.3). Additionally, the resource files also stored in the resource folder are distributed on all available Storage Peers existing in the respective SIPCA Peer Space.

Also the one to three job type interests of the Worker Peers are chosen randomly from the available job types stored in the resource folder. Furthermore, they receive a computational power between 1 and 5.
Moreover, a Client Peer creates the given number of query and job requests. Therefore, the demanded file of a query request is chosen randomly from the files stored in the resource folder. The job type of a job request is chosen the same way, whereas its minimal required computational power ranges from 1 to 4.5.
All entries containing this information are serialized, such that they can be reused in another benchmark with non-random initialization. The condition for this to work is that the same SIPCA Peer Space instance, identified by its name, is started with the same amount of Worker, Storage and Client Peers.

4.6.3 Resource Folder

The resource folder is an important part of the SIPCA framework, since it contains resources used for query requests and results, as well as information for peer initialization. Thus, the resource folder needs to contain a `topicList.txt` and a `jobTypeList.txt` file, both containing a list of keywords. Those files are used to simulate the topics and job type interests for Storage and Worker Peers. Furthermore, the resources folder contains files having their related topic somewhere in their file name. This way the files can be associated to their topic by the framework, as the classification of resources is not within the scope of this thesis.

4.6.4 Recluster Storage Peers

This parameter was introduced due to the fact, that no churn is simulated during the benchmarks (cf. section 6.1). Therefore, this parameter can be set true. In this case, each time a Storage Peer cannot answer a query and there is no suitable peer in its neighbor hood, the reclustering of Storage Peers can be triggered. This is possible every 5 seconds and only twice per query request, in order to avoid an infinite loop during a benchmark.

4.7 Summary

In this Chapter, first, a short overview of the Peer Model Java implementation [8] and an extension made to it is given. After a presentation of the framework's services, important classes and interfaces of the SIPCA framework are introduced.
Additionally, the metrics used to evaluate the benchmark are presented. Afterwards, the framework execution and configuration is explained. While there are many in-depth configurations offered for advanced users, for the majority of configuration parameters default values are given. Therefore, new users are able to directly focus on benchmarking.

CHAPTER 5

Swarm-Inspired Algorithms for Peer Clustering

Swarm-based algorithms are inspired by nature and aim at imitating the behavior of life forms organized in swarms and apply it to the solution of problems. The most widely used group of swarm-inspired clustering algorithms are ant-based algorithms, closely followed by bee-inspired algorithms. Similar to peer-to-peer systems, swarms are also perceived to be decentralized and self-organized systems. Thus, swarm-inspired algorithms are likely to provide satisfying results [19].
This Chapter is focused on defining the resources used in this thesis and the description of the swarm-inspired algorithms used for peer clustering. Thus, after a general, abstract explanation of an algorithm, its mapping to the peer clustering problem is discussed in order to clarify its implementation. These mappings are done in the course of this thesis. Furthermore, the criteria for clustering are introduced.

5.1 P2P Resource

As mentioned in section 4.6.2, P2P file resources are stored by Storage Peers and requested via queries by Client Peers. Such a resource in the P2P system can be defined as a combination of content and its metadata. For reasons of simplicity, only the file name is taken into account for the formal definition. Therefore, a *file resource* is defined as a singleton $r_f = (x_r)$, where x is the name of the file represented by the resource. A *query request* is modeled exactly the same way, namely $q_f = (x_q)$. Thus, a query is fulfilled if and only if $x_r \equiv x_q$, where r stands for resource and q stands for query.
Sample instances of a request and two file resources are shown below:

Query Request: ("filename.txt")
Matching File Resource: ("filename.txt")
Non-Matching File Resource: ("file.txt")

In contrast to a file resource, a job is no concrete resource. It is a metaphorical concept, defined as a combination of a job type, a descriptive categorization of the job, and a computational power. Nevertheless, it is an important part of the framework as it represents a Worker Peer, and is therefore defined as a tuple $r_j = (x_1, x_2)$, where x_1 depicts the job categories processed by a Worker Peer and x_2 is the currently available computational power of the Worker Peer, i.e. the available resources. Thus, x_1 is defined as an n-tuple of strings $x_1^r = (s_1, s_2, ..., s_n)$, where r stands for resource, $n \in \mathbb{N}$ and s_i is a string. A *job request* is structured in the same manner, with the difference that x_1 is a singleton $x_1^q = (s)$ representing one job category, where q stands for query, and x_2 represents the minimal required computational power. Therefore, a Worker Peer can only accept a job request if and only if $x_1^q \in x_1^r \wedge x_2^r \geq x_2^q$.
In order to clarify the formalism above, an illustrative example is provided by presenting sample instances of a job resource and three job requests.

Job Resource: ("sorting, optimization", 3.2)
Acceptable Job Request: ("sorting", 2.3)
Non-Acceptable Job Request: ("hash value calculation", 2.3)
Non-Acceptable Job Request: ("optimization", 3.3)

5.2 Clustering Criteria

As indicated by section 5.1, different criteria are used for clustering Storage and Worker Peers.
For Storage Peers, the main criterion is the topic interest, already mentioned in section 4.6.2 and, for reasons of simplicity, reflected by the file name of the respective file resource. The more topic interests two peers share, the higher is the probability of sharing the same cluster. For reasons of clarification an example is given below.

Storage Peer 1: ("sorting, optimization")
Storage Peer 2: ("sorting")
Storage Peer 3: ("hash value calculation")

In the example, Storage Peer 1 and Storage Peer 2 would be sharing a cluster, due to the fact that they have a topic interest in common, whereas Storage Peer 3 would be placed in another cluster.

On the contrary, for Worker Peers there are two main criteria for clustering: the job types processed by a Worker Peer and its currently available computational power. Thus, both of these criteria are taken into account in equal proportions when examining whether two peers are similar to each other or not. The below given example will illustrate this.

Worker Peer 1: ("sorting, optimization", 4.4)
Worker Peer 2: ("sorting", 4.3)
Worker Peer 3: ("sorting", 1.2)

In the example given above, Worker Peer 1 and Worker Peer 2 are very likely to share a cluster. Although Worker Peer 3 also shares the same job type, its probability of being part of the same cluster as Worker Peer 1 and Worker Peer 2 is not that high as its computational power is considerably lower.

5.2.1 Distance Measurement

Distance measurement is important for peer clustering, amongst other things, in order to learn which cluster is most suitable for a certain peer. Thus, the measurement of how well a peer fits into a cluster, in most algorithms called fitness value, is based on distance measurement. All measurements are based on the Euclidean distance formula equation (5.1), which measures the distance between two points p and q in an n-dimensional space [42].

$$d(p,q) = d(q,p) = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2} \tag{5.1}$$

The smaller the distance is, the more similar are two measurement objects. Vice versa, the larger the distance between two measurement objects is, the more dissimilar they are.

This thesis, in fact, uses the following ways of measuring, where the measurement type is chosen by the preference of the respective algorithm if there is given any.
To quantify the dissimilarity of strings, a variation of the Levenshtein distance [31] is used. The Levenshtein distance of strings as well as the numerical distance are expressed as a percentage, where the latter is based on equation (5.2) [50].

$$pd(p,q) = \frac{|p-q|}{\frac{p+q}{2}} \tag{5.2}$$

Peer-to-peer distance is used to calculate the distance between two Node Peers. An example of how exactly the calculation is performed is provided below.

Worker Peer 1 (p): ("sorting, optimization", 4.4)
Worker Peer 2 (q): ("sorting", 1.2)

$$\begin{aligned} d(p,q) &= \sqrt{(p_1-q_1)^2+(p_2-q_2)^2} \\ &= \sqrt{(levenshtein(\text{"sorting, optimization", "sorting"}))^2+(\frac{4.4-1.2}{\frac{4.4+1.2}{2}})^2} \\ &= \sqrt{(0.5)^2+(1.14)^2} \\ &= 1.24 \end{aligned}$$

Peer-to-cluster distance is used to calculate the distance between a Node Peer and a cluster mean. A cluster mean is embodied by its three most frequent keyword occurrences combined with their frequency of occurrence and the average available computational power. The calculation of the distance between a Node Peer and a cluster mean is illustrated below.

Cluster (p): ("sorting: 2, optimization: 1", 4.4)
Worker Peer (q): ("sorting", 1.2)

$$\begin{aligned} d(p,q) &= \sqrt{(p_1-q_1)^2+(p_2-q_2)^2} \\ &= \sqrt{(levenshtein(\text{"sorting, sorting, optimization", "sorting"}))^2+(\frac{4.4-1.2}{\frac{4.4+1.2}{2}})^2} \\ &= \sqrt{(0.33)^2+(1.14)^2} \\ &= 1.19 \end{aligned}$$

cluster-to-cluster distance is used to calculate the distance between two clusters. This calculation is illustrated by the example given below.

Cluster (p): ("sorting: 2, optimization: 1", 4.4)
Cluster (q): ("validity: 1", 1.2)

$$\begin{aligned} d(p,q) &= \sqrt{(p_1-q_1)^2+(p_2-q_2)^2} \\ &= \sqrt{(levenshtein(\text{"sorting, sorting, optimization", "validity"}))^2+(\frac{4.4-1.2}{\frac{4.4+1.2}{2}})^2} \\ &= \sqrt{(1)^2+(1.14)^2} \\ &= 1.52 \end{aligned}$$

5.2.1.1 Jaccard distance

Additionally to the distance measurement based on the Euclidean distance combined with the Levenshtein distance of strings, all types of distance measures were also implemented

using the Jaccard distance, based on the Jaccard index. Although the Euclidean distance is the default choice for the framework, this way an alternative way of distance measurement is provided. Furthermore, as the distance measurements are, just as the peer clustering algorithms, implemented in a Plug & Play manner, they easily interchangeable.
Also in this type of distance measurement, the numerical distance is expressed as a percentage (cf. equation (5.2)) The formula for the Jaccard distance is given in equation (5.3).

$$J_\delta(p,q) = 1 - \frac{p \cap q}{p \cup q} \tag{5.3}$$

To demonstrate the calculation of the peer-to-peer distance using the Jaccard distance measurement, an example is given below:

Worker Peer 1 (p): ("sorting, optimization", 4.4)
Worker Peer 2 (q): ("sorting", 1.2)

$$\begin{aligned}
J_\delta(p,q) &= \frac{J_\delta(p_1,q_1) + pd(p_2,q_2)}{2}\\
&= \frac{1 - \frac{\{\text{sorting, optimization}\}\cap\{\text{sorting}\}}{\{\text{sorting, optimization}\}\cup\{\text{sorting}\}} + \frac{|4.4-1.2|}{\frac{4.4+2.2}{2}}}{2}\\
&= \frac{1 - \frac{\{\text{sorting}\}}{\{\text{sorting, optimization}\}} + 1.14}{2}\\
&= \frac{1 - \frac{1}{2} + 1.14}{2}\\
&= \frac{0.5 + 1.14}{2}\\
&= 0.82
\end{aligned}$$

The calculation of the peer-to-cluster distance using the Jaccard distance is illustrated by the example given below:

Cluster (p): ("sorting: 2, optimization: 1", 4.4)
Worker Peer (q): ("sorting", 1.2)

$$
\begin{aligned}
J_\delta(p,q) &= \frac{J_\delta(p_1,q_1)+pd(p_2,q_2)}{2}\\
&= \frac{1-\frac{\{\text{sorting, optimization}\}\cap\{\text{sorting}\}}{\{\text{sorting, optimization}\}\cup\{\text{sorting}\}}+\frac{|4.4-1.2|}{\frac{4.4+2.2}{2}}}{2}\\
&= \frac{1-\frac{\{\text{sorting}\}}{\{\text{sorting, optimization}\}}+1.14}{2}\\
&= \frac{1-\frac{1}{2}+1.14}{2}\\
&= \frac{0.5+1.14}{2}\\
&= 0.82
\end{aligned}
$$

An example how the cluster-to-cluster calculation is performed is provided below:

Cluster (p): ("sorting: 2, optimization: 1", 4.4)
Cluster (q): ("validity: 1", 1.2)

$$
\begin{aligned}
J_\delta(p,q) &= \frac{J_\delta(p_1,q_1)+pd(p_2,q_2)}{2}\\
&= \frac{1-\frac{\{\text{sorting, optimization}\}\cap\{\text{validity}\}}{\{\text{sorting, optimization}\}\cup\{\text{validity}\}}+\frac{|4.4-1.2|}{\frac{4.4+2.2}{2}}}{2}\\
&= \frac{1-\frac{\{\}}{\{\text{sorting, optimization, validity}\}}+1.14}{2}\\
&= \frac{1-\frac{0}{3}+1.14}{2}\\
&= \frac{1+1.14}{2}\\
&= 1.07
\end{aligned}
$$

5.3 Artificial Bee Colony (ABC)

The Artificial Bee Colony (ABC) algorithm [22] is based on the foraging behavior of honey bees. A possible solution is represented by a food source and the food source's nectar amount complies with the fitness or quality of the associated solution. In an artificial bee hive the foraging tasks are divided between employed bees and onlooker bees. Employed bees go to the food sources visited by them before, whereas an onlooker bee waits for the employed bee, giving them quality information about the visited food source. Then, the onlooker bee chooses a food source to exploit on the basis of the nectar

quality. The more nectar a food source offers, the larger is the probability to be chosen by an onlooker bee.
In the ABC algorithm the number of employed bees and onlooker bees equals the number of food sources. Therefore, each food source is visited by only one employed bee.
The procedure of the ABC algorithm is illustrated in algorithm 5.1 [22].

Algorithm 5.1: Artificial Bee Colony algorithm

```
   Input: number of clusters $k$, number of food sources $noSources$, maximum
          number of iterations $maxIteration$
 1 initialization;
 2 for $j := 1$ to $maxIteration$ do
 3     for each employed bee do
 4         produce new solution $v_i$;
 5         calculate fitness value;
 6         apply greedy selection;
 7     end
 8     calculate the probability values $p_i$ for the solutions;
 9     for each onlooker bee do
10         select a solution depending on $p_i$;
11         produce new solution $v_i$;
12         calculate fitness value;
13         apply greedy selection;
14     end
15     memorize the best solution so far;
16 end
```

In each iteration step, each employed bee produces a modification of the current solution v_i depending on the local information [22]:

$$v_i = z_i + \phi(z_i - z_k) \tag{5.4}$$

where z_i is the current solution and z_k is a randomly selected food source differing from z_i. ϕ is a random number between -1 and 1.
After producing the new source its nectar amount is tested. Therefore, the fitness of the solution is calculated the following way [22]:

$$fit_i = \frac{1}{1 + f_i} \tag{5.5}$$

where f_i is the sum on all instances of Euclidean distance between an instance and the associated cluster center, divided by the number of instances.
A greedy selection is then applied to the newly produced solution and the current one, where the better one is kept in memory.

After all employed bees have completed the search, the nectar information of the food sources is shared with the onlooker bees. Therefore, the probability value p_i has to be calculated for each solution:

$$p_i = \frac{fit_i}{\sum_{n=1}^{SN} fit_n} \tag{5.6}$$

where SN is the number of food sources equalling the number of employed bees and onlooker bees.
Based on this probability value, each onlooker bee chooses one food source. The higher the probability value of a certain solution is, the higher is the chance to be chosen by an onlooker bee. Thus, if a solution has a very high probability value, it may be chosen by multiple onlooker bees.
The onlooker bee then produces a new solution based on the selected food source using equation (5.4). The fitness value of this solution is computed by equation (5.5) and, just as in the employed bees phase, a greedy selection process is applied between the newly produced solution and the probabilistically chosen one.

The richest food source shall be memorized across all iterations.

The algorithm is mapped to the peer clustering problem the following way: At the initialization the given numbers of food sources $noSources$, i.e. solutions to the clustering problem, is created. This happens by randomly assigning the peers to be clustered to a given number of clusters k. This is done in such a way that $\frac{n}{k}$ peers are assigned to each cluster, where n is the number of peers to be clustered. Then, the rest of the peers are assigned to a randomly chosen cluster.
Afterwards, in each iteration the following procedure is performed. Each employed bee creates a new solution based on its own food source and a randomly chosen one. More specifically, for each peer in both solutions the associated cluster is expressed as a numeric value. This makes it possible to insert these numeric values into equation (5.4), resulting in a cluster expressed as a numeric value which shall be associated to the corresponding peer. Thus, the composition of the newly created solution is such that this is done for each peer. Next, the fitness value for the bee's solution and new solution is calculated as described above. This also applies to the choice of the better solution and the calculation of the probability value p_i for each food source.
Eventually, based on the probability value, each onlooker bee chooses a solution. This means, the higher the fitness value of a solution is, the higher is p_i, and consequently the higher is the probability to be chosen by an onlooker bee. Then, the onlooker bee's procedure is basically the same as the employed bee's.

The basic characteristics of the ABC algorithm are presented in table 5.1 [38].

Mechanism of exploration	Random initialization.
Mechanism of exploitation	Employed and onlooker bees searching the neighborhood.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of k non-empty clusters inhabited by all nodes in the considered network.

Table 5.1: Basic characteristics of ABC.

5.4 Combination of Artificial Bee Colony Algorithm and K-Means (ABCK)

The ABCK algorithm [2] is basically a combination of the procedure of the Artificial Bee Colony algorithm and classical K-Means. According to Armano and Farmani [2], the K-Means algorithm is highly dependent on the centroid initialization while being computationally light, whereas the ABC algorithm is rather time-consuming. Therefore, a combination of those two algorithms shall complement each other.
The setting of the ABCK algorithm is basically the same as of the ABC algorithm, regarding employed bees, onlooker bees and the food sources. The procedure differs from the original one by adding an additional K-Means step, being described in algorithm 5.2 [2].

Every time a bee produced a modification of a solution v_i, in the employed bee phase as well as in the onlooker bee phase, the K-Means algorithm is applied on this solution. Then, the solution is evaluated by means of a criterion called distortion, introduced by Armano and Farmani [2] and defined as follows:

$$E = \sum_{k=1}^{K} \sum_{z_i \epsilon C_k} \|z_i - C_k\|^2 \tag{5.7}$$

where z_i is the ith instance belonging into cluster C_k. The distortion sums up the squared distance between all peers and their associated cluster centers, using the peer-to-cluster distance mentioned in section 5.2.1. If the distortion is low, the clustering is good, as it measures the inner distances within each cluster and thus, the distortion shall be minimized. Therefore, the solution having the smaller distortion is kept in memory. Also in this procedure, the best solution shall be memorized across all iterations.

Due to the fact that this algorithm is very similar to the pure ABC algorithm, the mapping to the peer clustering problem is exactly the same as described above (cf. section 5.3), with the only exception that each newly produced solution is locally optimized by applying K-Means on it.

The basic characteristics of the ABCK algorithm are presented in table 5.2 [38].

Algorithm 5.2: Artificial Bee Colony algorithm combined with K-Means

```
Input: number of clusters $k$, number of food sources $noSources$, maximum
       number of iterations $maxIteration$
1  initialization;
2  for $j := 1$ to $maxIteration$ do
3      for each employed bee do
4          produce new solution $v_i$;
5          apply k-means on $v_i$;
6          calculate distortion;
7          apply greedy selection;
8      end
9      calculate the probability values $p_i$ for the solutions;
10     for each onlooker bee do
11         select a solution depending on $p_i$;
12         produce new solution $v_i$;
13         apply k-means on $v_i$;
14         calculate distortion;
15         apply greedy selection;
16     end
17     memorize the best solution so far;
18 end
```

Mechanism of exploration	Random initialization.
Mechanism of exploitation	Employed and onlooker bees searching the neighborhood. Application of K-Means in the onlooker bee's phase.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of k non-empty clusters inhabited by all nodes in the considered network.

Table 5.2: Basic characteristics of ABCK.

5.5 Ant-based Clustering

In Ant-based Clustering [34] the ants act on a two-dimensional grid which is populated randomly with items. In this algorithm an item represents the object which shall be clustered. Thus, in the scope of this thesis an item equates to a Node Peer.
In each iteration, an ant is selected randomly. If there is an item at the ant's current position and the ant is not carrying one, the selected ant can pick the item. If the location is free, a carried item can be dropped by the ant. The probability of picking and dropping, respectively, is influenced by the similarity of the peers placed in the surrounding area, clarified in equation (5.10). This procedure is illustrated in algorithm 5.3 [34].

Algorithm 5.3: Algorithm for ant-based clustering

```
Input: step size of an ant $stepsize$, maximum number of iterations $maxIteration$,
       picking constant $k_{pick}$, dropping constant $k_{drop}$, scaling parameter $\alpha$
1  randomly distribute items on the grid;
2  randomly place ants on the grid;
3  for $i := 1$ to $maxIteration$ do
4      choose ant randomly;
5      move ant randomly by $stepsize$;
6      if $antCarriesItem \wedge cellIsEmpty$ then
7          $p_{drop}$ := calculate drop probability;
8          if $randomDouble \leq p_{drop}$ then
9              drop item;
10         end
11     end
12     else if $\neg antCarriesItem \wedge \neg cellIsEmpty$ then
13         $p_{pick}$ := calculate pick probability;
14         if $randomDouble \leq p_{pick}$ then
15             pick item;
16         end
17     end
18 end
```

The picking and dropping probabilities for a given position i are calculated the following way [34]:

$$p_{pick}(i) = (\frac{k_{pick}}{k_{pick} + f(i)})^2 \tag{5.8}$$

and

$$p_{drop}(i) = \begin{cases} 2f(i) & \text{if} f(i) < k_{drop}, \\ 1 & \text{otherwise} \end{cases} \tag{5.9}$$

where k_{pick} and k_{drop} are constants and $f(i)$ is the neighborhood function of the current location i:

$$f(i) = \begin{cases} \frac{1}{d^2} \sum_j (1 - \frac{d(i,j)}{\alpha}) & \text{if} f(i) > 0, \\ 0 & \text{otherwise} \end{cases} \tag{5.10}$$

where the sum extends over all locations j in the surrounding area of location i and $d(i, j)$ is the measurement of dissimilarity between the items located at cells i and j, and therefore computed by using the peer-to-peer distance (cf. section 5.2.1). α is a scaling constant and d^2 equals the size of the local neighborhood of i to normalize the result.

This algorithm is mapped to the peer clustering problem as follows. Firstly, Node Peers and ants are randomly distributed on the grid. Then, the ants move the Node Peers as described above, by picking them up and dropping them at another location. At the end of the algorithm, a cluster is formed by all Node Peers which are horizontally, vertically or diagonally adjacent to each other.

The basic characteristics of the Ant-based Clustering algorithm are presented in table 5.3 [38].

Mechanism of exploration	Random position initialization of ants and items.
Mechanism of exploitation	Ants picking and dropping items.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of non-empty clusters inhabited by all nodes in the considered network.

Table 5.3: Basic characteristics of Ant-based Clustering.

5.6 Ant K-Means

The Ant K-Means algorithm [30] is based on combining ant colony optimization with conventional K-Means clustering. It is inspired by the natural behavior of ant colonies, as they leave a trail of pheromones to communicate with each other. Pheromones are left by an ant following a certain path. Thus, the more ants take the same trail, the more pheromones lay on the trail. Consequently, this particular path becomes more attractive for other ants and the shortest route can be obtained.
Thus, the conventional K-Means is modified in such a way, that it utilizes the described behavior. The procedure of the Ant K-Means algorithm is illustrated in algorithm 5.4 [30].

In each iteration the pheromones of each path have to be updated, using the following formula by [30]:

$$\tau_{ij} = (1-p)\tau_{ij} + \frac{Q}{TWCV} \tag{5.11}$$

where p is the pheromone decay parameter, Q is a constant and $TWCV$ is the total within cluster variance, the sum of the squared distance of each peer to its respective cluster mean.
Then, each ant m assigns each peer i to a cluster mean j with a probability P_{ij}^m which is calculated as follows [30]:

$$P_{ij}^m = \frac{\tau_{ij}^\alpha \eta_{ij}^\beta}{\sum_c^k \tau_{ic}^\alpha \eta_{ic}^\beta} \tag{5.12}$$

where α is the relative importance of the trail and β is the relative importance of the visibility η, which is the inverse of the distance between peer i and cluster center j, i.e. $\eta_{ij} = \frac{1}{d_{ij}}$.
Thereafter, the cluster means are updated and the new $TWCV$ has to be calculated. This procedure is repeated until the $TWCV$ does not change anymore. If the new $TWCV$ is smaller than the global best $TWCV$, it is replaced.

Algorithm 5.4: Algorithm for Ant K-Means

```
Input: number of clusters $k$, number of ants $noAnts$, maximum number of
       iterations $maxIteration$, pheromone decay parameter $p$, relative
       importance of the trail $\alpha$, relative importance of the visibility $\beta$, constant
       $Q$
1  initialization;
2  lay equal pheromone on each path;
3  for $l := 1$ to $maxIteration$ do
4      for $m := 1$ to $noAnts$ do
5          while $TWCV$ changed do
6              update pheromones $\tau_{ij}$;
7              assign each object to a cluster with probability $P_{ij}^m$;
8              update cluster centers;
9              calculate $TWCV$;
10         end
11         if $TWCV < bestTWCV$ then
12             $bestTWCV :=$ TWCV;
13         end
14     end
15     perturbation;
16 end
```

Next, a step called perturbation is performed in order to not to get stuck with a local minimum. Basically, in this step every ant is newly initialized, just as described in the initialization step below.

The algorithm is mapped to the peer clustering problem the following way: At the initialization step the given number of ants *noAnts* is created. As each ant holds its own solution to the peer clustering problem, this includes the initialization of k cluster means by randomly assigning peers to it. Afterwards, lay equal pheromone on each path. Then, in each iteration step, the pheromones of each path are updated and each peer is assigned to the cluster with the highest probability P_{ij}^m, as described above.

The basic characteristics of the Ant K-Means algorithm are presented in table 5.4 [38].

Mechanism of exploration	Random initialization. Perturbation.
Mechanism of exploitation	Evaporation and usage of pheromone trails.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of k non-empty clusters inhabited by all nodes in the considered network.

Table 5.4: Basic characteristics of Ant K-Means.

5.7 Particle Swarm Optimization (PSO)

In Particle Swarm Optimization [37] an individual grouped into a swarm is referred to as a particle. A swarm can be seen as a flock of birds flying towards an optimum. Therefore, a particle searches for the best solution, using the best position encountered by itself and its swarm. Thus, a wide area can be searched while heading towards an optimum.
Each particle i represents a complete solution of the clustering problem and has to maintain the information of its current position x_i, its velocity v_i and its personal best position y_i, where the personal best position is the position where the particle obtained the highest fitness value so far. Additionally to the personal best position there is also a global best position $\hat{y}$, determined from the entire swarm. The procedure of the PSO algorithm is described in algorithm 5.5 [37].

Algorithm 5.5: Algorithm for Particle Swarm Optimization

```
Input: number of clusters $k$, number of particles $noParticles$, maximum number
       of iterations $maxIteration$, constant $w_1$, constant $w_2$
1  initialization;
2  for $j := 1$ to $maxIteration$ do
3      for $i := 1$ to $noParticles$ do
4          update velocity $v_i$;
5          update position $x_i$;
6          if $f(x_i) < f(y_i)$ then
7              $y_i := x_i$
8          end
9          if $f(x_i) < f(\hat{y})$ then
10             $\hat{y} := x_i$
11         end
12     end
13 end
```

In each iteration t, the velocity and position of a particle is updated using the following formulae by [37]:

$$v_i(t+1) = wv_i(t) + c_1 r_1(t)(y_i(t) - x_i(t)) + c_2 r_2(t)(\hat{y}(t) - x_i(t)) \tag{5.13}$$

and

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{5.14}$$

where w is the inertia weight, c_1 and c_1 are constants and r_1 and r_1 are random values between 0 and 1.
The particle's fitness at the current position is computed by means of a fitness function f encapsulating the characteristics of the optimization problem and thus, reflecting the optimality of the solution. Then, the personal best position is updated by comparing the fitness value of the current position to the fitness value of the incumbent personal best

position. If the fitness value of the current position is better, the incumbent personal best position is replaced by the current one.

This algorithm is mapped to the peer clustering problem as follows. At the initialization step the given number of clusters k is initialized by randomly assigning one peer to it, representing the cluster mean.
Then, in each iteration the solution of each particle has to be adapted. This means, each peer of the solution is assigned to the best suiting (i.e. nearest) cluster. Afterwards, the fitness of the current solution is calculated. This is done by using the ratio between the average intra-cluster distance (i.e. peer-to-cluster distance) and the average inter-cluster distance (i.e. cluster-to-cluster distance).
If the fitness of the current solution is smaller than the fitness of the personal and global best solution, respectively, the respective best solution is replaced by the current one. Subsequently, the centroids of each solution have to be updated. This complies with the above mentioned step of updating the velocity and position of a particle. In order to have a non-numeric procedure which, at the same time, is in accordance with the formulas of equation (5.13) and equation (5.14), this is done the following way. The average of the cluster means of the current solution and the personal best solution is calculated, as well as of the cluster means of the current solution and the global best solution. Next, the two results of this operation are averaged again, resulting in the new cluster center. This way the new position of a particle is always influenced by the personal and global best solution, aiming at optimizing the current solution.

The basic characteristics of the PSO algorithm are presented in table 5.5 [38].

Mechanism of exploration	Random initialization.
Mechanism of exploitation	Generation of a new solution based on the current, the personal best and the global best solution.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of k non-empty clusters inhabited by all nodes in the considered network.

Table 5.5: Basic characteristics of PSO.

5.8 Slime Mold

Šešum-Čavić et. al. present a swarm intelligent algorithm in [45], which imitates the life cycle of the Dictyostelium discoideum slime mold, and is based on the Dictyostelium discoideum algorithm for numerical optimization proposed in [36]. In this section an adaption of the Slime Mold algorithm for peer clustering is presented, which is based on the two mentioned works.

The slime mold Dictyostelium discoideum (Dd) is a social collective of amoebae, using spores to reproduce. The Dd's life cycle goes through five stages, depicted in figure 5.1:

vegetative movement, aggregation, mound, slug movement, and spore dispersal [24].

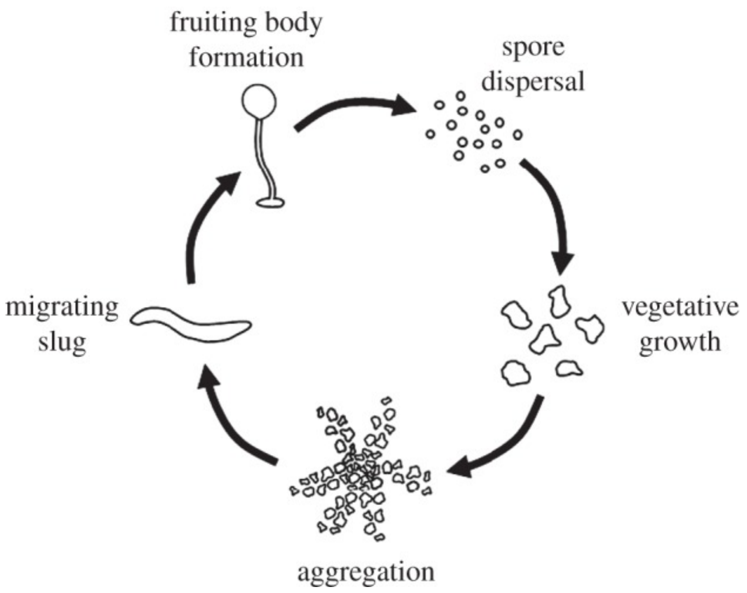

Figure 5.1: Life cycle of the Dictyostelium discoideum slime mold [20].

At the first stage, vegetative movement, amoebae are searching for food. An amoeba begins to starve if there is not enough food supply available. Thus, an amoeba remains in this state until the food supply diminishes.
At this point the aggregation stage begins, where amoebae show the first attempt of cooperative behavior. In this stage there exist two types of amoebae, namely pacemakers and aggregating amoebae. A chemical called *cyclic Adenosine Monophosphate* (cAMP) is emitted by a pacemaker. The aggregating amoebae aggregate towards pacemakers as they are attracted by cAMP. By this aggregation amoebae are forming a mound.
At this stage, the amoebae organize themselves into two groups, prestalk and prespore, based on their fitness level, i.e. how well-fed they are. The prestalk amoebae will be part of the the fruiting body and, therefore, eventually die, whereas the prespore amoebae will be dispersed as spores.
Together, the two groups form a slug, divided into a head, composed of prestalk amoebae, and a tail, formed by prespore amoebae. The slug moves into a direction where the possibility of a culmination, like a light source, exists.
Then, at the last stage, it starts to form a fruiting body, where the prestalk amoebae become part of the fruiting body's stalk and will eventually die. The prespore amoebae are dispersed as spores, such that the Dd's life cyle can start again by entering the vegetative stage [24].

The Slime Mold algorithm is mapped to the peer clustering problem the following way: At the first stage of the life cycle, the given numbers of amoebae *noAmoebae* is initialized. As every amoeba holds its own solution of the clustering problem, this is done by randomly choosing k cluster means.

Algorithm 5.6: Algorithm for Slime Mold

```
Input: number of clusters $k$, number of amoebae $noAmoebae$, maximum number
       of iterations $maxIteration$, restriction parameter $\varepsilon$
for $j := 1$ to $maxIteration$ do
    Procedure vegetativeMovement():
        initialization of amoebae;
    end
    Procedure aggregation():
        for $i := 1$ to $noAmoebae$ do
            while $|TWCV - newTWCV| > \varepsilon$ do
                assign each object to nearest cluster center;
                update cluster centers;
                calculate new TWCV;
            end
        end
    end
    Procedure mound():
        divide amoebae into $prespore$ and $prestalk$;
    end
    Procedure slugMovement():
        for $i := 1$ to $noPrespore$ do
            combine $solution_i$ with $solution_{best}$;
        end
    end
    Procedure dispersion():
        remember global best solution $solution_{best}$;
    end
end
```

In the second stage, aggregation, the clustering begins by assigning each peer to the nearest cluster center, representing the role of a pacemaker. Afterwards, the cluster means are updated and the new total within cluster variation (TWCV) is calculated. This is repeated until the old TWCV and the new TWCV converge to a limit given by ε. Then, in the mound stage, the one half of the amoebae having the better TWCV is categorized as prespore, the other half as prestalk.
At the next stage, slug movement, the solution of each prespore amoeba is combined with the global best solution the same way as described in algorithm 5.5, by averaging the two cluster means.
At the dispersion stage, the global best solution is remembered by selecting the amoeba having the smallest TWCV.
In the next iteration of the algorithm, the prespore amoebae are kept, whereas the prestalk amoebae are replaced by newly initialized amoebae.

The basic characteristics of the Slime Mold algorithm are presented in table 5.6 [38].

Mechanism of exploration	Vegetative movement.
Mechanism of exploitation	Aggregation. Slug movement.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of k non-empty clusters inhabited by all nodes in the considered network.

Table 5.6: Basic characteristics of Slime Mold.

5.9 Slime Mold K-Means

The Slime Mold K-Means algorithm for clustering is based on the same biological approach as the pure Slime Mold for clustering (cf. section 5.8). Therefore, the setting is basically the same, but the procedure differs from the original one by adding an additional K-Means step, which is described in algorithm 5.7.

The mapping to the peer clustering problem of the Slime Mold K-Means algorithm is very similar to the Slime Mold mapping. However, there are two deviations:

In the aggregation stage, each peer is assigned to the nearest cluster center, which represents the role of the pacemaker. After that, there is no more local optimization taking place.
In return, at the end of the dispersion stage, the solutions of all prespore amoebae are locally optimized by applying K-Means.

The basic characteristics of the Slime Mold K-Means algorithm are presented in table 5.7 [38].

Algorithm 5.7: Algorithm for Slime Mold K-Means

```
Input: number of clusters $k$, number of amoebae $noAmoebae$, maximum number
       of iterations $maxIteration$
for $j := 1$ to $maxIteration$ do
    Procedure vegetativeMovement():
        initialization of amoebae;
    end
    Procedure aggregation():
        for $i := 1$ to $noAmoebae$ do
            aggregate towards nearest pacemaker;
        end
    end
    Procedure mound():
        divide amoebae into $prespore$ and $prestalk$;
    end
    Procedure slugMovement():
        for $i := 1$ to $noPrespore$ do
            combine $solution_i$ with $solution_{best}$;
        end
    end
    Procedure dispersion():
        remember global best solution $solution_{best}$;
        for $i := 1$ to $noPrespore$ do
            apply k-means;
        end
    end
end
```

Mechanism of exploration	Vegetative movement.
Mechanism of exploitation	Slug movement. Dispersion.
Feasibility of solution	The feasible region is the space of all candidate solutions, i.e. the set of k non-empty clusters inhabited by all nodes in the considered network.

Table 5.7: Basic characteristics of Slime Mold K-Means.

5.10 Summary

At the beginning of this Chapter the P2P resources and clustering criteria are discussed. In the course of this, also the methods of distance measurement used by the framework are presented.
Afterwards, multiple swarm-inspired clustering algorithms are discussed in detail and it

is explained how they are mapped to the peer clustering problem. Such algorithms can be inspired by a variety of organisms organizing in swarms, such as ants or bees, whereas the Slime Mold algorithm [45], for example, imitates the life-cycle of the Dictyostelium discoideum slime mold.

CHAPTER 6

Evaluation

In this Chapter the Slime Mold and Slime Mold K-Means peer clustering algorithm adaptions are analyzed and compared to the nine intelligent and non-intelligent clustering algorithms, which were introduced in prior Chapters of this work.

6.1 Benchmark Methodology

For the subsequent comparison, the Slime Mold and Slime Mold K-Means peer clustering algorithms, presented in section 5.8 and section 5.9, are benchmarked alongside Artificial Bee Colony, Artificial Bee Colony combined with K-Means, Ant-based Clustering, Ant K-Means, Fuzzy C-Means, Genetic K-Means, Hierarchical Clustering, K-Means and Particle Swarm Optimization. All these algorithms are briefly described in section 2.3.

All benchmarks are carried out on Google Cloud Platform's Compute Engine [47]. In particular, a "n1-standard-16" instance was used, which includes 16 vCPUs and 60 GB RAM. According to [48], the following processors are available for n1 machine types: 2.0 GHz Intel Xeon Scalable Processor, 2.2 GHz Intel Xeon E5 v4, 2.3 GHz Intel Xeon E5 v3, 2.5 GHz Intel Xeon E5 v2 and 2.6 GHz Intel Xeon E5.

The general benchmark setups are the following:

- **P2P network nodes:** The amount of the nodes is varied in three levels: low (L_n), medium (M_n), high (H_n), where $L_n = 50$, $M_n = 100$ and $H_n = 200$. This means for example, that in a network of medium size 100 Storage Peers, 100 Worker Peers and 100 Client Peers are attendant. Furthermore, it shall be noted, that no churn is simulated. Thus, there is no fluctuation of node participation.
  rewrite
- **Requests:** For the Sensitivity Analysis the amount of requests per Client Peer is varied in three levels, also: low (L_r), medium (M_r), high (H_r), where $L_r = 1$,

$M_r = 5$ and $H_r = 10$ query and job requests per Client Peer.
On the contrary, all competitive benchmarks are executed with exactly three queries and job requests per Client Peer, as the requests have shown to have no direct impact on the performance of the peer clustering algorithms.

- **ReclusterStorages:** As no churn is simulated during the benchmarks, the reclusterStorages parameter is set to true, in order to have multiple Storage Peer clustering results per benchmark execution. For a more detailed explanation see section 4.6.4.
- **Benchmark execution:** Each configuration of an algorithm is executed ten times, in order to have a variety of different node initializations as described in detail in section 4.6.2. Thus, in the result data the average metric values are recorded.

The suggested default values, mentioned in section 4.6, are used for the rest of the framework parameters.

The metrics used for the evaluation are execution time, the Davies-Bouldin index (DBI), the Dunn index (DI), the silhouette coefficient (SC) and Averaged Dissimilarity coefficient (ADC). All metrics are described in detail in section 4.4.
The metric of execution time measures the performance of the algorithms which can be important, especially in time critical scenarios. In order to examine the quality of the peer clustering solutions from different perspectives, three established and well-known metrics are chosen. While the Davies-Bouldin index [11] uses artificial cluster means to calculate the ratio of the within cluster cohesion to the between cluster separation, the other two measures use only the distances between individual nodes. Furthermore, the Dunn index [14, 15] uses the worst case scenarios, i.e. the global minimum distance between two clusters and the global maximum distance between any pair of peers within the same cluster, to calculate the ratio between cohesion and separation, whereas the Davies-Bouldin index and the Silhouette coefficient [39, 23] are calculating with average values. In contrast to the other two metrics which directly calculate the ratio between cohesion and separation, the Silhouette coefficient normalizes the result by subtracting the average within-cluster distance from the minimum between-cluster distance and dividing the result by the greater of both numbers.
Additionally, one simple metric, the Averaged Dissimilarity coefficient (cf. section 4.4.5), is introduced in the course of this thesis. It mainly distinguishes from the other metrics as it only focuses on the cohesion within the clusters, while neglecting the well-separation of the clusters.

6.2 Sensitivity Analysis

The goal of the Sensitivity Analysis is to find the best combination of parameters for each benchmarked algorithm. This parameter tuning is performed on the parameter's value range recommended by the author of the particular algorithm or determined in preliminary benchmarks. Thus, a parameter is only subject of the Sensitivity Analysis

if no precise value is recommended by the author. As for the most benchmarked algorithms the number of clusters has to be parameterized, a global range for this value is used, depending on the number of nodes in the network, i.e. number of clusters = $\{n*0.1; n*0.2; n*0.3; n*0.4; n*0.5\}$, where n is the number of nodes in the network. These values were used for the following reasons: As it makes a great difference for the quality of the clustering solution whether 10 or 100 peers are clustered into 3 clusters, the number of clusters shall be dependent on the number of nodes. Furthermore, the upper limit for k is set to $\frac{n}{2}$, as otherwise a balanced distribution would be harder to accomplish. As the number of clusters is required in nearly each benchmarked algorithm it can be considered to be one of the most important parameters and therefore, five values are tested, fairly distributed between 10 % and 50 % of the number of nodes to be clustered.
The number of iterations *maxIteration* and the number of agents, like *noAnts* and *noAmoebae*, are also parameters occurring more often, therefore for them three global values are specified to be tested, in case there does not exist an expert recommendation. For the number of agents the minimum and maximum recommended values from other algorithms are used as lower and upper limit (10 from PSO and 50 from Genetik K-Means), whereas the middle value is the average. Thus, the number of agents are tested for the values 10, 30 and 50. For the number of iterations the lower limit is also defined by the minimum recommended value (20 from ABCK), whereas the upper limit is set to 100 as it is a value twice recommended (from Genetik K-Means and PSO) and due to performance reasons. The middle value is set to 50. Thus, *maxIteration* is tested for the values 20, 50, 100. Also other non-recommended parameters are tested with three values. In detail, for all peer clustering algorithm parameters being subject of the Sensitivity Analysis the optimal parameter value is determined based on the metrics mentioned in section 6.1 for all combinations of network sizes (L_n, M_n) and request levels (L_r, M_r, H_r). These optimal values are then used in the competitive benchmarks.

ABC As many other algorithms, the ABC peer clustering algorithm, described in section 5.3, requires a given number of clusters k, in which to cluster the nodes. Therefore, as mentioned before, a global range is used which depends on the number of nodes n participating in the network, i.e. the number of objects to be clustered. Therefore, the value of k ranges between $\frac{n}{10}$ and $\frac{n}{2}$. The values of all other parameters used for the competitive benchmark are based on the recommendations in [22].
The parameter values and ranges, respectively, are shown in table 6.1, whereas the results of the Sensitivity Analysis are depicted in table 6.2.

As can be seen in table 6.2, for each combination of network sizes and request levels, the value of k performed best with $k = n*0.5$. Therefore, in a network consisting of 100 nodes, $k = 50$.

ABCK Also for the peer clustering algorithm ABCK, which was presented in detail in section 5.4, the number of clusters k has to be determined. The other parameter values

Parameter	**Value (Range)**	**Source**
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks
$noSources$	20	[22]
$maxIteration$	1000	[22]

Table 6.1: ABC parameter values before Sensitivity Analysis.

Nodes n	**Requests**	k
50	1	$n * 0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6.2: ABC Sensitivity Analysis results.

of the algorithm are recommended by [2].
The parameter values and value ranges are listed in table 6.3.

Parameter	**Value (Range)**	**Source**
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks
$noSources$	10	[2]
$maxIteration$	20	[2]

Table 6.3: ABCK parameter values before Sensitivity Analysis.

In contrast to ABC, the ABCK algorithm performs best with a value of $k = n * 0.4$ for each combination setup, which is stated in table 6.4. It may be noticeable that the values for $noSources$ and $maxIteration$ differ for ABC and ABCK, although the algorithms are rather similar. This is, as it was decided to use recommended values for each algorithm if available.

Ant-based Clustering The Ant-based Clustering algorithm, described in section 5.5, is the only benchmarked algorithm which does not require a parameterized number of clusters. The values for $maxIteration$, k_{pick}, k_{drop} and $stepsize$ are suggested by [5], [18] and [34], respectively. The only parameter to be determined through Sensitivity Analysis is the scaling parameter α. α shall be ranged between 0 and 1, and was therefore tested

Nodes n	**Requests**	$\boldsymbol{k}$
50	1	$n * 0.4$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6.4: ABCK Sensitivity Analysis results.

with the values given below.
The parameter values and value ranges before the Sensitivity Analysis, respectively, can be seen in table 6.5, whereas its results are shown in table 6.6.

Parameter	**Value (Range)**	**Source**
$stepsize$	1	[34]
$maxIteration$	$max(1000000; \sqrt{2000 * n})$	[5]
k_{pick}	0.1	[18]
k_{drop}	0.1	[18]
α	0.3, 0.6, 0.9	preliminary benchmarks

Table 6.5: Ant-based Clustering parameter values before Sensitivity Analysis.

As presented in table 6.6, for each combination of network sizes and request level, the value of $\alpha = 0.3$ outperformed the other inspected values for the scaling parameter α.

Nodes n	**Requests**	$\boldsymbol{\alpha}$
50	1	0.3
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6.6: Ant-based Clustering Sensitivity Analysis results.

Ant K-Means The Ant K-Means algorithm, which was described in detail in section 5.6, has three parameter values to be determined, namely k, $noAnts$ and $maxIteration$. The

values for the parameters p, α, β and Q are recommended by [30].
The parameter values and value ranges are shown in table 6.7.

Parameter	Value (Range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks
$noAnts$	10, 30, 50	preliminary benchmarks
$maxIteration$	20, 50, 100	preliminary benchmarks
p	0.9	[30]
α	0.5	[30]
β	1	[30]
Q	1	[30]

Table 6.7: Ant K-Means parameter values before Sensitivity Analysis.

The results of the Sensitivity Analysis can be seen in table 6.8. It illustrates that the values $k = n * 0.1$, $noAnts = 10$ and $maxIteration = 20$ performed best across all combination setups.

Nodes n	Requests	k	$noAnts$	$maxIteration$
50	1	$n * 0.1$	10	20
50	5			
50	10			
100	1			
100	5			
100	10			

Table 6.8: Ant K-Means Sensitivity Analysis results.

Fuzzy C-Means Also for the Fuzzy C-Means algorithm, shorty described in section 2.3.1.3, the number of clusters c has to be determined. According to [4], a fuzzifier m chosen between 1.5 and 3.0 gives good results for most data. The value for the $acceptanceBorder$ is suggested by [7].
The parameter values and value ranges, respectively, can be seen in table 6.9, whereas the Sensitivity Analysis results are shown in table 6.10.

As illustrated in table 6.10, for each combination of network sizes and request levels, the values of $c = n * 0.1$ and $m = 1.5$ performed best.

Parameter	Value (Range)	Source
c	$n*0.1$, $n*0.2$, $n*0.3$, $n*0.4$, $n*0.5$	preliminary benchmarks
m	1.5, 2.2, 3.0	[4]
$acceptanceBorder$	$\frac{1}{c}$	[7]

Table 6.9: Fuzzy C-Means parameter values before Sensitivity Analysis.

Nodes n	Requests	c	m
50	1	$n*0.1$	1.5
50	5		
50	10		
100	1		
100	5		
100	10		

Table 6.10: Fuzzy C-Means Sensitivity Analysis results.

Genetic K-Means The Genetic K-means algorithm was shortly described in section 2.3.2.5. Most of its parameter values are recommended by [26]. Only the values of the number of clusters k and the $alleleModifier$ have to be determined by the Sensitivity Analysis. According to [26], $alleleModifier \geq 1$, therefore, its lower limit is 1, whereas the upper limit is set to 10 due to preliminary benchmarks. Then, the natural number 5 is chosen for the middle value, as it is about the mean.
Table 6.11 shows the values and value ranges, respectively, while the results of the Sensitivity Analysis can be seen in table 6.12.

Parameter	Value (Range)	Source
k	$n*0.1$, $n*0.2$, $n*0.3$, $n*0.4$, $n*0.5$	preliminary benchmarks
$maxGeneration$	100	[26]
$nrOfChromosomes$	50	[26]
c	2	[26]
$alleleModifier$	1, 5, 10	preliminary benchmarks
$mutationProbability$	0.05	[26]

Table 6.11: Genetic K-Means parameter values before Sensitivity Analysis.

The Genetic K-Means algorithm performs best with a value of $k = n*0.1$ and $alleleModifier = 10$ for each combination setup. This indicates that the algorithm performs better when the probability for mutation is higher.

Nodes n	**Requests**	k	$alleleModifier$
50	1	$n * 0.1$	10
50	5		
50	10		
100	1		
100	5		
100	10		

Table 6.12: Genetic K-Means Sensitivity Analysis results.

Hierarchical Clustering Also for the Hierarchical Clustering algorithm, which was shortly described in section 2.3.1.1, the number of cluster k has to be determined. The parameter value range before the Sensitivity Analysis is shown in table 6.13, whereas its results can be inspected in table 6.14.

Parameter	**Value (Range)**	**Source**
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks

Table 6.13: Hierarchical Clustering parameter values before Sensitivity Analysis.

Across all combination setups the Hierarchical Clustering algorithm performs best with a value of $k = n * 0.5$, shown in table 6.14.

Nodes n	**Requests**	k
50	1	$n * 0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6.14: Hierarchical Clustering Sensitivity Analysis results.

K-Means The K-Means algorithm as well only has the number of clusters parameter k to be determined by the Sensitivity Analysis.

Therefore, table 6.15 lists the value range, whereas table 6.16 shows the Sensitivity Analysis results.

Parameter	Value (Range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks

Table 6.15: K-Means parameter values before Sensitivity Analysis.

The Sensitivity Analysis shows that K-Means also performs best with a parameter value of $k = n * 0.5$.

Nodes n	Requests	k
50	1	$n * 0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6.16: K-Means Sensitivity Analysis results.

PSO Also for the PSO peer clustering algorithm, presented in detail in section 5.7, only the number of clusters parameter k has to be determined, as all other parameter values used for the competitive benchmark are recommended by [37].
The parameter value and value ranges before the Sensitivity Analysis, respectively, are listed in table 6.17, while its results are presented in table 6.18.

Parameter	Value (Range)	Source
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks
$noParticles$	10	[37]
$maxIteration$	100	[37]
w_1	0.5	[37]
w_2	0.5	[37]

Table 6.17: PSO parameter values before Sensitivity Analysis.

As can be seen in table 6.18, for each combination of network sizes and request levels, the PSO algorithm's value of k performs best with $k = n * 0.5$.

Nodes n	**Requests**	**k**
50	1	$n * 0.5$
50	5	
50	10	
100	1	
100	5	
100	10	

Table 6.18: PSO Sensitivity Analysis results.

Slime Mold The Slime Mold peer clustering algorithm, which was described in detail in section 5.8, has multiple parameter values to be investigated by the Sensitivity analysis. Those parameter value ranges are presented in table 6.19. The threshold value ε has to be carefully chosen, as it is important for the local optimization. If the lower limit is chosen too small, the algorithm could stuck at local optimization at the expense of performance, therefore it is set to 0.3. If the upper limit is chosen too big, the local optimization ends too early which may affect the quality of the clustering solution negatively. Therefore, the upper limit is set to 0.9, whereas the mean 0.6 is taken for the middle value. The results of the Sensitivity Analysis can be seen in table 6.20.

Parameter	**Value (Range)**	**Source**
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks
$noAmoebae$	10, 30, 50	preliminary benchmarks
$maxIteration$	20, 50, 100	preliminary benchmarks
ε	0.3, 0.6, 0.9	preliminary benchmarks

Table 6.19: Slime Mold parameter values before Sensitivity Analysis.

Also the Slime Mold algorithm performs best with the same values for all combination of network sizes and request levels, i.e. $k = n * 0.5$, $noAmoebae = 10$, $maxIteration = 20$ and $\varepsilon = 0.6$.

Slime Mold K-Means Just like the Slime Mold algorithm, all of the Slime Mold K-Means algorithm's parameter values have to be determined. Slime Mold K-Means was presented in detail in section 5.9.
The parameter values and value ranges are listed in table 6.21.

Nodes n	**Requests**	k	$noAmoebae$	$maxIteration$	ε
50	1	$n * 0.5$	10	20	0.6
50	5				
50	10				
100	1				
100	5				
100	10				

Table 6.20: Slime Mold Sensitivity Analysis results.

Parameter	**Value (Range)**	**Source**
k	$n * 0.1$, $n * 0.2$, $n * 0.3$, $n * 0.4$, $n * 0.5$	preliminary benchmarks
$noAmoebae$	10, 30, 50	preliminary benchmarks
$maxIteration$	20, 50, 100	preliminary benchmarks

Table 6.21: Slime Mold K-Means parameter values before Sensitivity Analysis.

The results of the Sensitivity Analysis can be seen in table 6.22. It shows that the values $k = n * 0.5$, $noAmoebae = 10$ and $maxIteration = 20$ performed best across all combinations of network size and request level.

Nodes n	**Requests**	k	$noAmoebae$	$maxIteration$
50	1	$n * 0.5$	10	20
50	5			
50	10			
100	1			
100	5			
100	10			

Table 6.22: Slime Mold K-Means Sensitivity Analysis results.

6.3 Raw Result Data

In the following tables the raw result data of the competitive benchmarks are shown: ABC in table 6.24, ABCK in table 6.25, Ant-based Clustering in table 6.26, Ant K-Means in table 6.27, Fuzzy C-Means in table 6.28, Genetic K-Means in table 6.29, Hierarchical

Clustering in table 6.30, K-Means in table 6.31, PSO in table 6.32, Slime Mold in table 6.33 and Slime Mold K-Means in table 6.34.

The tables contain the average result for each network size and shows separate results for Worker Peer clustering, Storage Peer clustering and the average of both. For reasons of representation, the metrics discussed above (cf. section 6.1) are abbreviated. Table 6.23 provides the corresponding explanations.

Abbreviation	Explanation
type	Specifies, whether the results are from Worker Peer clustering, Storage Peer clustering or the average of both.
nodes	Number of Worker Peer, Storage Peer and Client Peer nodes participating in the benchmark.
time	Average execution time of one algorithm execution in seconds. Rounded to 2 decimals.
DBI	Average Davies-Bouldin index of one clustering result. Rounded to 4 decimals.
DI	Average Dunn index of one clustering result. Rounded to 4 decimals.
SC	Average Silhouette coefficient of one clustering result. Rounded to 4 decimals.
ADC	Average Averaged Dissimilarity coefficient of one clustering result. Rounded to 4 decimals.

Table 6.23: Metric abbreviation explained for competitive benchmark.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	25.20 *sec*	1.5053	0.2619	0.0662	0.6160
	100	85.14 *sec*	1.7040	0.2287	0.0037	0.7146
	200	278.20 *sec*	1.6696	0.2247	-0.0458	0.8456
storage	50	25.77 *sec*	1.4112	0.4041	0.0429	0.5350
	100	87.53 *sec*	1.5507	0.3165	-0.0261	0.6021
	200	263.67 *sec*	1.6591	0.4167	-0.1685	0.9457
all	50	25.48 *sec*	1.4583	0.3330	0.0545	0.5755
	100	86.34 *sec*	1.6274	0.2726	-0.0112	0.6583
	200	270.93 *sec*	1.6643	0.3207	-0.1072	0.8957

Table 6.24: ABC raw result data.

type	**Nodes**	**time**	**DBI**	**DI**	**SC**	**ADC**
worker	50	2.10 *sec*	1.1258	0.3534	0.2230	0.6745
	100	8.27 *sec*	1.1541	0.3273	0.2102	0.6581
	200	37.53 *sec*	1.1948	0.3233	0.1979	0.6403
storage	50	1.87 *sec*	1.0744	0.6046	0.3326	0.4196
	100	7.81 *sec*	1.1352	0.5528	0.3207	0.4499
	200	36.61 *sec*	1.2013	0.5086	0.2908	0.4471
all	50	1.98 *sec*	1.1001	0.4790	0.2778	0.5470
	100	8.04 *sec*	1.1447	0.4400	0.2654	0.5540
	200	37.07 *sec*	1.1980	0.4160	0.2443	0.5396

Table 6.25: ABCK raw result data.

type	**Nodes**	**time**	**DBI**	**DI**	**SC**	**ADC**
worker	50	0.51 *sec*	2.2532	0.3566	-0.0826	1.2113
	100	0.55 *sec*	2.3191	0.3311	-0.1141	1.2045
	200	0.57 *sec*	2.3400	0.3061	-0.1303	1.1831
storage	50	1.17 *sec*	1.9799	0.5667	-0.0443	0.9695
	100	2.01 *sec*	2.0073	0.7000	-0.0640	0.9593
	200	2.11 *sec*	2.1104	0.6000	-0.0655	0.9437
all	50	0.84 *sec*	2.1165	0.4617	-0.0634	1.0986
	100	0.79 *sec*	2.1632	0.5156	-0.0891	1.1002
	200	0.83 *sec*	2.2252	0.4531	-0.0979	1.0802

Table 6.26: Ant-based Clustering raw result data.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	6.18 *sec*	1.5582	0.4922	0.0711	1.0180
	100	66.94 *sec*	1.6239	0.4426	0.0512	1.0069
	200	524.95 *sec*	1.7113	0.5143	0.0324	0.9760
storage	50	5.19 *sec*	1.5636	1.0000	0.0798	0.8721
	100	42.82 *sec*	1.6277	0.9539	0.0704	0.8941
	200	482.40 *sec*	1.6542	0.9333	0.0567	0.9176
all	50	5.68 *sec*	1.5609	0.7461	0.0755	0.9472
	100	54.88 *sec*	1.6258	0.6982	0.0608	0.9531
	200	503.68 *sec*	1.6828	0.7238	0.0445	0.9407

Table 6.27: Ant K-Means raw result data.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	0.02 *sec*	-	0.0182	-0.0789	1.1150
	100	0.09 *sec*	-	0.0041	-0.0462	1.1076
	200	0.79 *sec*	-	0.0015	0.0135	1.0900
storage	50	0.09 *sec*	-	0.7500	0.000	0.9696
	100	0.29 *sec*	-	0.5333	0.000	0.9877
	200	1.68 *sec*	-	0.5667	0.000	0.9940
all	50	0.05 *sec*	-	0.3841	-0.0395	1.0423
	100	0.19 *sec*	-	0.2687	-0.0231	1.0476
	200	1.23 *sec*	-	0.2841	0.0068	1.0420

Table 6.28: Fuzzy C-Means raw result data.

type	**Nodes**	**time**	**DBI**	**DI**	**SC**	**ADC**
worker	50	10.38 *sec*	1.5802	0.3919	0.0773	0.8903
	100	35.31 *sec*	1.5587	0.3918	0.0767	0.8950
	200	138.773 *sec*	1.5644	0.3876	0.0707	0.8877
storage	50	10.14 *sec*	1.6621	0.7243	0.1535	0.7294
	100	38.46 *sec*	1.6271	0.7113	0.1576	0.7627
	200	139.21 *sec*	1.6046	0.7124	0.1436	0.7441
all	50	10.26 *sec*	1.6212	0.5581	0.1154	0.8099
	100	36.89 *sec*	1.5929	0.5515	0.1172	0.8289
	200	138.99 *sec*	1.5845	0.5500	0.1071	0.8159

Table 6.29: Genetic K-Means raw result data.

type	**Nodes**	**time**	**DBI**	**DI**	**SC**	**ADC**
worker	50	0.14 *sec*	0.8281	0.3958	0.4112	0.3437
	100	0.67 *sec*	0.8781	0.3698	0.4158	0.3194
	200	3.86 *sec*	0.9201	0.3490	0.4166	0.3359
storage	50	0.14 *sec*	0.8965	0.6504	0.4558	0.2547
	100	0.70 *sec*	0.9485	0.6155	0.4557	0.2835
	200	4.22 *sec*	0.9489	0.5729	0.4294	0.2858
all	50	0.14 *sec*	0.8623	0.5231	0.4335	0.2992
	100	0.68 *sec*	0.9133	0.4927	0.4358	0.3015
	200	4.04 *sec*	0.9345	0.4610	0.4230	0.3108

Table 6.30: Hierarchical Clustering raw result data.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	0.01 *sec*	1.0458	0.2785	0.2777	0.5110
	100	0.03 *sec*	1.0726	0.2300	0.2770	0.5056
	200	27.02 *sec*	1.1965	0.3663	0.2494	0.4940
storage	50	0.01 *sec*	0.9889	0.4886	0.3791	0.3080
	100	0.04 *sec*	1.0378	0.4637	0.3825	0.3363
	200	27.56 *sec*	1.2395	0.6393	0.3107	0.3279
all	50	0.01 *sec*	1.0173	0.3835	0.3284	0.4095
	100	0.04 *sec*	1.0552	0.3469	0.3298	0.4210
	200	27.29 *sec*	1.2180	0.5028	0.2801	0.4109

Table 6.31: K-Means raw result data.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	4.07 *sec*	0.9781	0.2625	0.2792	0.4929
	100	17.34 *sec*	1.0523	0.1916	0.2670	0.5047
	200	70.54 *sec*	1.1410	0.1282	0.2559	0.5131
storage	50	4.00 *sec*	0.9397	0.4874	0.3780	0.3060
	100	17.29 *sec*	1.0413	0.4699	0.3766	0.3284
	200	72.33 *sec*	1.1066	0.4671	0.3580	0.3067
all	50	4.04 *sec*	0.9589	0.3749	0.3286	0.3994
	100	17.31 *sec*	1.0468	0.3307	0.3218	0.4165
	200	71.43 *sec*	1.1238	0.2977	0.3069	0.4099

Table 6.32: PSO raw result data.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	0.79 *sec*	1.2406	0.3168	0.1701	0.6459
	100	2.93 *sec*	1.2816	0.3035	0.1596	0.6467
	200	11.39 *sec*	1.3102	0.2678	0.1599	0.6403
storage	50	0.81 *sec*	1.2235	0.5935	0.2416	0.5136
	100	2.97 *sec*	1.2519	0.5462	0.2379	0.5450
	200	11.55 *sec*	1.2716	0.4995	0.2190	0.5215
all	50	0.80 *sec*	1.2320	0.4552	0.2058	0.5798
	100	2.95 *sec*	1.2667	0.4249	0.1988	0.5959
	200	11.47 *sec*	1.2909	0.3837	0.1895	0.5809

Table 6.33: Slime Mold raw result data.

type	Nodes	time	DBI	DI	SC	ADC
worker	50	0.92 *sec*	1.2091	0.3210	0.1933	0.6436
	100	3.88 *sec*	1.2328	0.2965	0.1893	0.6362
	200	16.67 *sec*	1.2422	0.2545	0.2067	0.6155
storage	50	0.87 *sec*	1.1828	0.6008	0.2678	0.5047
	100	3.61 *sec*	1.1815	0.5531	0.2792	0.5258
	200	16.01 *sec*	1.2160	0.5186	0.2487	0.5216
all	50	0.89 *sec*	1.1960	0.4609	0.2305	0.5742
	100	3.75 *sec*	1.2071	0.4248	0.2342	0.5810
	200	16.34 *sec*	1.2291	0.3865	0.2277	0.5685

Table 6.34: Slime Mold K-Means raw result data.

6.4 Competitive Analysis

In this section, the result data of the competitive benchmarks showed above (cf. section 6.3) is evaluated.

There is one peer clustering algorithm that falls out of line regarding the Davies-Bouldin index and the Dunn index. The Fuzzy C-Means algorithm is the only benchmarked algorithm which allows peers to be in multiple clusters at the same time. As both, the Davies-Bouldin index and the Dunn index, measure the separation between clusters, the Fuzzy C-Means algorithm shows extraordinary high results for the Davies-Bouldin index and rather volatile results for the Dunn index.

In figure 6.1 the execution time for all benchmarked algorithms is depicted. Due to the fact, that five algorithms have a shorter execution time than 20 seconds considering each network size level, figure 6.2 provides a closer look into these algorithms.
Those five algorithms, Fuzzy C-Means, Ant-based Clustering, Hierarchical Clustering, Slime Mold and Slime Mold K-Means, are quite close to each other. The Ant-based Clustering algorithm shows a very constant execution time for each network size level. This effect is due to the fact that it is the only benchmarked algorithm which does not depend on the number of peers to be clustered. Its complexity is only dependent on the number of iterations and the number of ant agents moving across the grid. The Ant-based Clustering algorithm is also the only one, where the execution time of Worker Peer clustering and Storage Peer clustering differs notably, as Storage Peer clustering takes about twice as long as Worker Peer clustering.
While Fuzzy C-Means stands out with a low execution time, it is more than 3 times faster than the Hierarchical Clustering algorithm, which may be due to its nature of having overlapping clusters. However, it increases by 3.8 times (L_n to M_n) and by 6.5 times (M_n to H_n), whereas Hierarchical Clustering's execution time increases by a factor of 4.9 (L_n to M_n) and a factor of 5.9 (M_n to H_n).
Furthermore, the Slime Mold algorithm outperforms the Slime Mold K-Means algorithm, which could be caused by the application of K-Means during the dispersion phase. As mentioned below, K-Means does not scale very well which could affect the Slime Mold K-Means algorithm. While they start with a comparable execution time at network size level L_n, Slime Mold is 21 % faster at level M_n and 30 % faster at level H_n. Thus, the execution time of Slime Mold increases by 3.7 times (L_n to M_n) and by 3.9 times (M_n to H_n), while the execution time of Slime Mold K-Means increases by a factor of 4.2 (L_n to M_n) and by a factor of 4.4 (M_n to H_n). Thus, the Slime Mold algorithm outperforms all other algorithms, except Ant-based Clustering, in terms of scalability related to execution time.
Interestingly, while K-Means scales rather good from network size level L_n to M_n, it then shows an increase by a factor of about 682 to level H_n, which may be due to the fact that the algorithm is repeated until no more cluster changes occur.

Consequently, six algorithms require more than 20 seconds for their execution, first and foremost Ant K-Means with an average execution time of 503.68 seconds for a network size of level H_n. Thus, in terms of execution time Ant K-Means scales definitely worst with a time increase of about ten times each network size level, as it is also repeated until no peer changes its cluster anymore.

As execution time is an important factor for clustering algorithms, but by far not the only one, four metrics measuring the quality of the peer clustering result are now discussed.

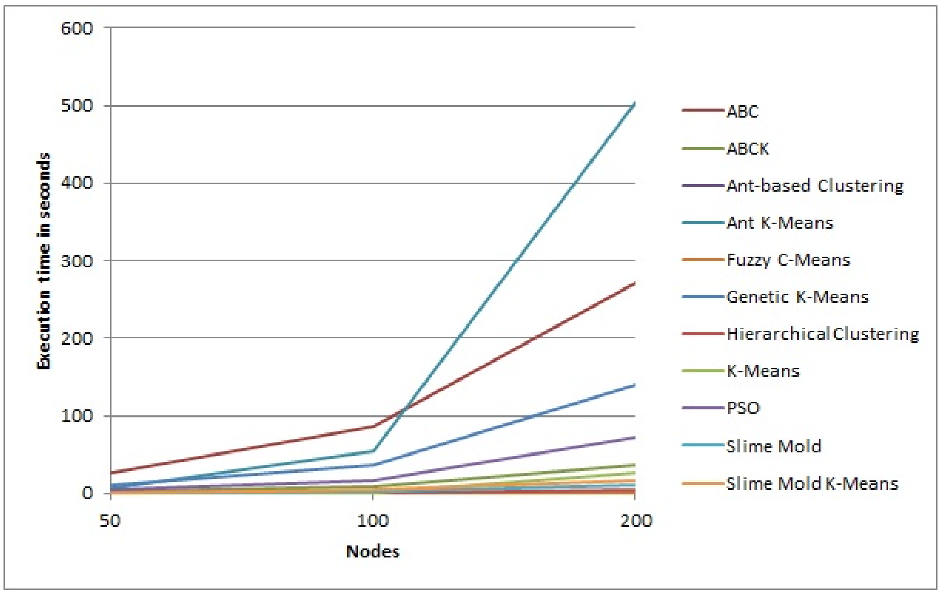

Figure 6.1: Execution time results for both Node Peer types and all network size levels.

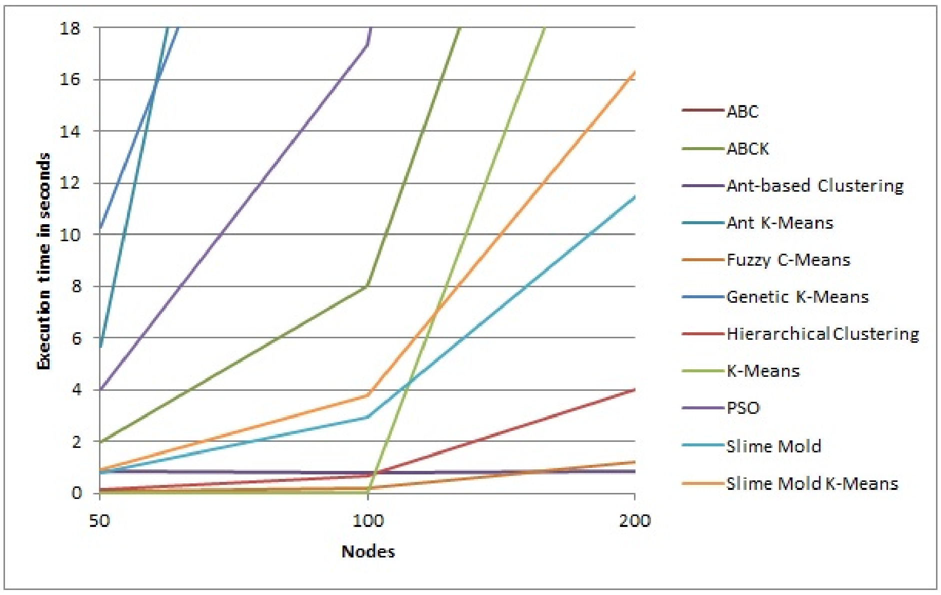

Figure 6.2: Execution time results zoomed in.

Figure 6.3 presents the overall results for the Davies-Bouldin index for all benchmarked algorithms. In figure 6.4 the comparison of the Davies-Bouldin index results between Worker and Storage Peer clustering is depicted for ABC, ABCK, Ant-based Clustering and Ant K-Means, whereas in figure 6.5 it is shown for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means.
As mentioned in section 4.4.2, the smaller the Davies-Bouldin index is, the better is the clustering result.

The Hierarchical Clustering algorithm shows the best results for the Davies-Bouldin index, including a low increase rate of 6 % (L_n to M_n) and 2 % (M_n to H_n). It is followed by PSO, K-Means and ABCK, where PSO and K-Means scale rather poorly in comparison. While ABCK increases by 4 % (L_n to M_n) and 5 % (M_n to H_n), PSO increases by 9 % and 7 % per network size level and K-Means even by 4 % and 15 %. This may be due to the fact that in Hierarchical Clustering, in contrast to the other algorithms, no random cluster initialization is made, but instead the two globally best matching clusters are merged together, starting with each peer having its own cluster.
Slime Mold K-Means and Slime Mold follow immediately, scaling fairly well with an increase of 1 % (L_n to M_n) and 2 % (M_n to H_n), and 3 % and 2 %, respectively. Thus, also with regard to the Davies-Bouldin index Slime Mold and Slime Mold K-Means may not show to perform best in absolute numbers during the benchmarks which were set up, but impress in terms of scalability regarding effectiveness.

The algorithm which definitely performed worst regarding the Davies-Bouldin index is the Ant-based Clustering algorithm. Also it is the algorithm which shows the most discrepancy between Worker Peer clustering and Storage Peer clustering, with Storage Peer clustering being 12 % (50 nodes), 13 % (100 nodes) and 10 % (200 nodes) more effective than Worker Peer clustering. This could be caused by the randomness of how the peers are distributed on the grid on which the ants move around.
Overall, Genetic K-Means and Hierarchical Clustering are the only benchmarked algorithms which show better results for Worker Peer clustering than for Storage Peer clustering.
Furthermore, K-Means performs better in Storage Peer clustering on network size levels L_n and M_n, but shows better results for Worker Peer clustering on a network consisting of 200 nodes.

In figure 6.6 the Dunn index results for all benchmarked algorithms are depicted. Figure 6.7 provides the comparison of the Dunn index results between Worker Peer clustering and Storage Peer clustering for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means, while figure 6.8 provides it for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means.
As mentioned in section 4.4.3, the higher the Dunn index is, the better is the clustering result.

By far, the Ant K-Means algorithm shows the best results for the average Dunn index, as it is 34 % (50 nodes), 27 % (100 nodes) and 32 % (200 nodes) more effective than its successor Genetic K-Means, indicating that Ant K-Means has less worst case scenario

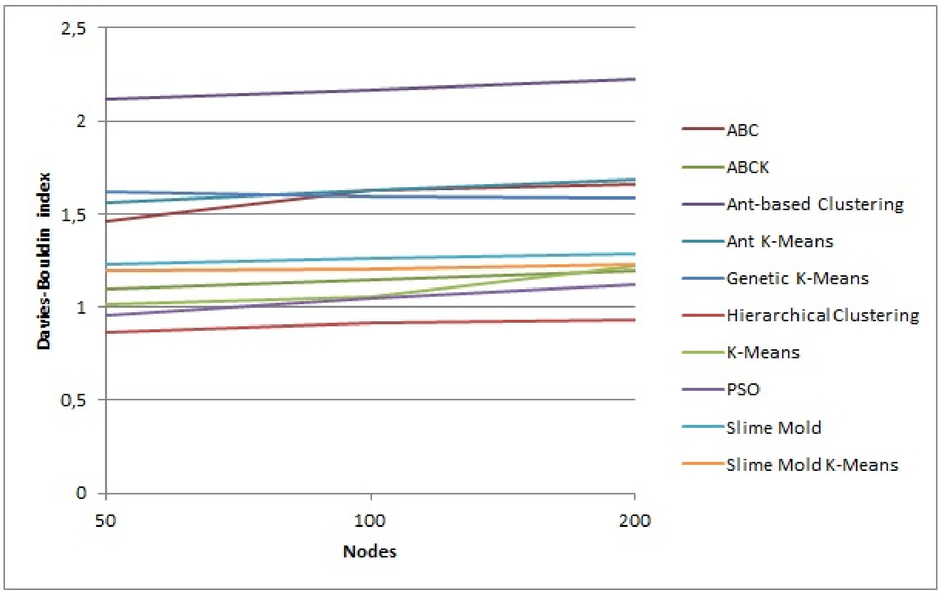

Figure 6.3: Davies-Bouldin results for both Node Peer types and all network size levels.

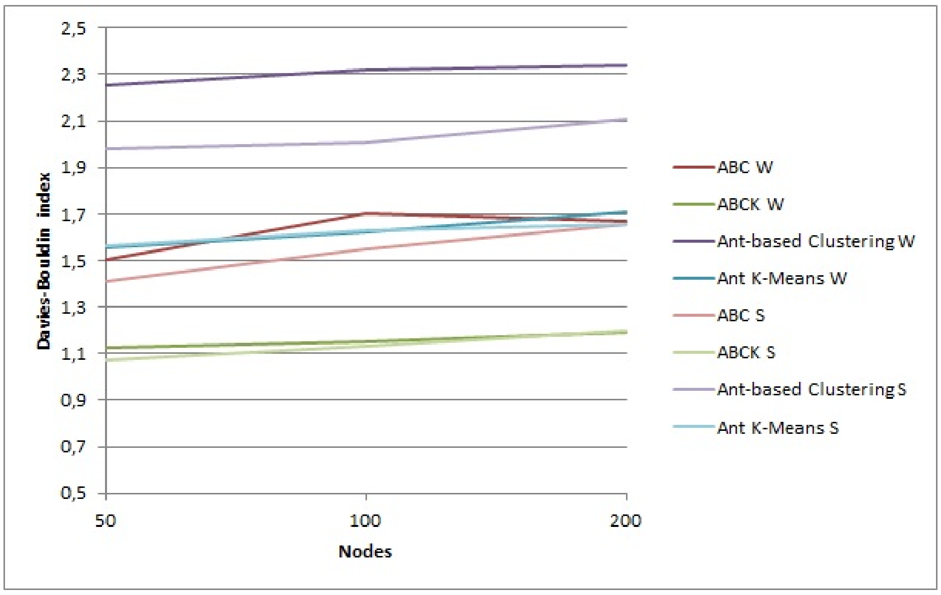

Figure 6.4: Comparison of Davies-Bouldin index results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering and Ant K-Means for all network size levels.

solutions, i.e low cohesion and well-separation. By contrast, Genetic K-Means has only an average decrease rate of 1 %, whereas Ant K-Means first decreases 6 % (L_n to M_n), but then again increases 4 % (M_n to H_n). This indicates, that Genetic K-Means is much

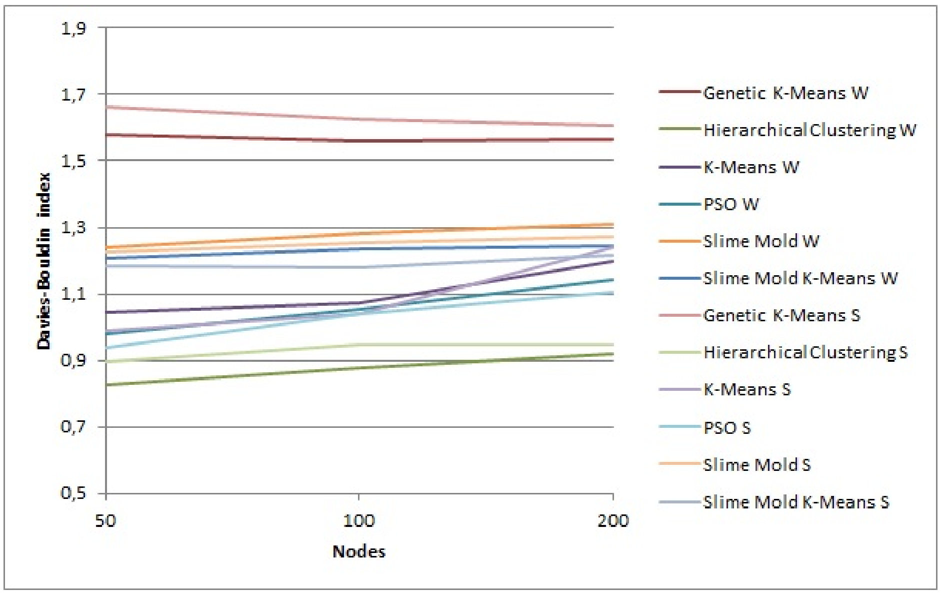

Figure 6.5: Comparison of Davies-Bouldin index results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels.

more stable than Ant K-Means.
Slime Mold and Slime Mold K-Means are midranged with very similar results for the Dunn index. They have a decrease around 7 % from network size level L_n to M_n and a decrease around 9 % from network size level M_n to H_n.
Fuzzy C-Means and ABC share the last rank, showing a similar unstable behavior as Ant K-Means.
Interestingly, K-Means first exhibits a decrease of 9 % (L_n to M_n), but then extraordinarily increases by 45 % (M_n to H_n).

Overall it can be said, that according to the Dunn index results every benchmarked algorithm performs by far better for Storage Peer clustering than for Worker Peer clustering.

Figure 6.9 presents the overall results for the Silhouette coefficient for all benchmarked algorithms. In figure 6.10 the comparison of the Silhouette coefficient results between Worker and Storage Peer clustering is depicted for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means, whereas in figure 6.11 it is shown for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means.
As mentioned in section 4.4.4, the higher the Silhouette coefficient is, the better is the clustering result.

Just like for the Davies-Bouldin index results, the Hierarchical Clustering algorithm also shows the best results for the Silhouette coefficient. It is about 32 % (50 and 100 nodes) and 51 % (200 nodes) more effective than K-Means, which follows right behind. However,

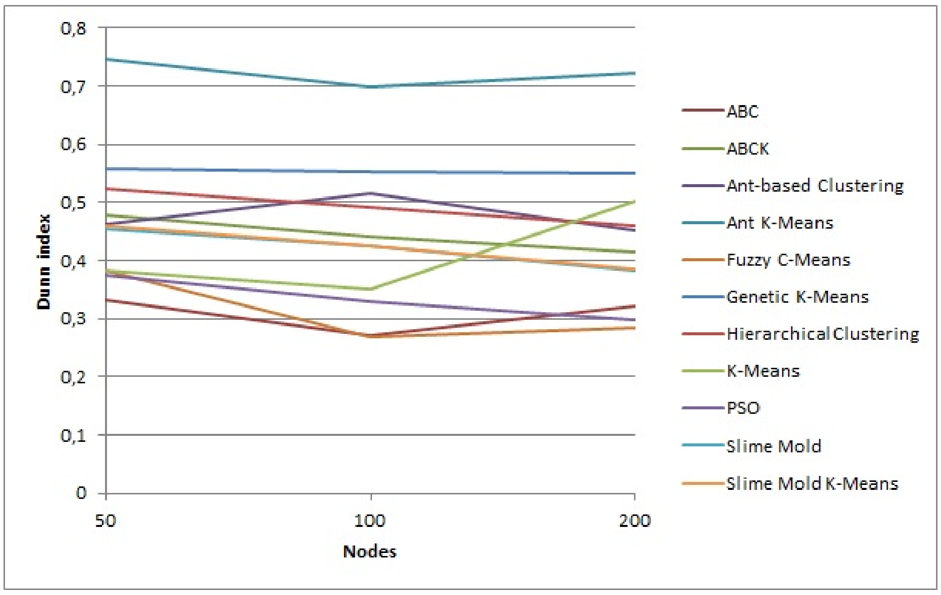

Figure 6.6: Dunn index results for both Node Peer types and all network size levels.

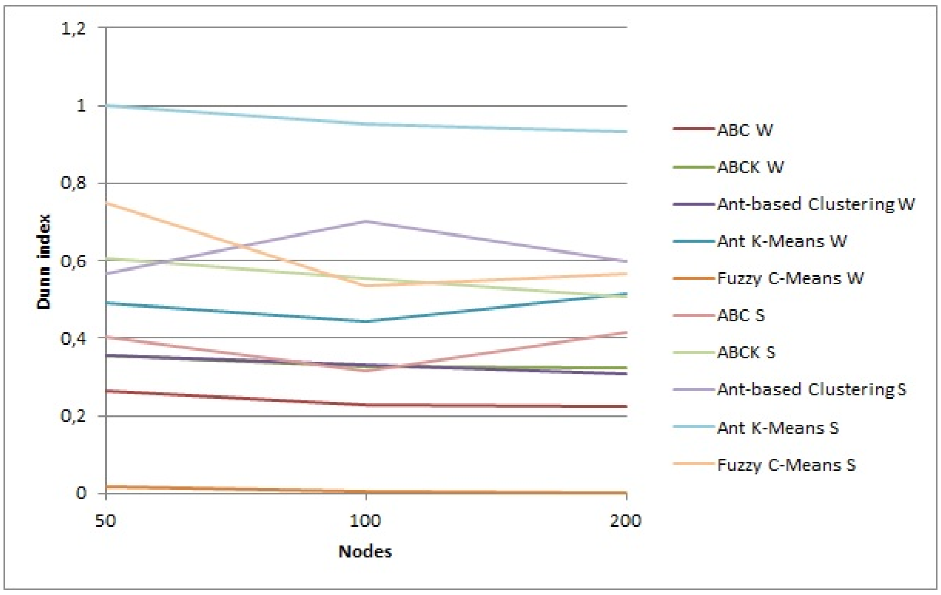

Figure 6.7: Comparison of Dunn index results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means for all network size levels.

Hierarchical clustering increases by 0.5 % (L_n to M_n) and then slightly decreases by 3 % (M_n to H_n), whereas K-Means has an increase of 0.4 % (L_n to M_n) and then a decrease of 15 % (M_n to H_n). The reason for the effectiveness of Hierarchical Clustering

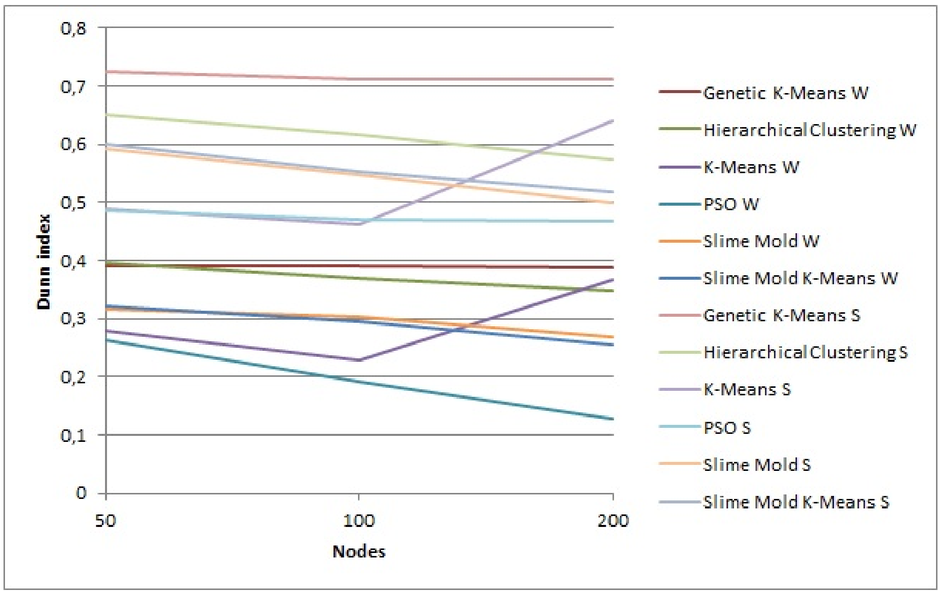

Figure 6.8: Comparison of Dunn index results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels.

is mentioned above and applies also here.
The Slime Mold and Slime Mold K-Means algorithms are again placed in the upper midrange, where Slime Mold K-Means outperforms Slime Mold by 12 % (50 nodes), 18 % (100 nodes) and 20 % (200 nodes). Thus, Slime Mold K-Means first slightly increases by about 2 % from network size level L_n to M_n, then decreases by 3 % from network size level M_n to H_n, while Slime Mold has a decrease of 3 % from network size level L_n to M_n and 5 % from network size level M_n to H_n. This, also considering the results of the Davies-Bouldin index and execution time, indicates, that the application of K-Means in the dispersion phase instead of the optimization by convergence to threshold value ε in the aggregation phase requires more time but seems to be more efficient.
As well as for the Davies-Bouldin index results, the Ant-based Clustering algorithm also performs worst regarding the Silhouette coefficient, which, as already mentioned, could be caused by its randomness.
While Fuzzy C-Means starts off with the second-worst rank, it then massively increases by 42 % (L_n to M_n) and 129 % (M_n to H_n). The contrary behavior shows the ABC algorithm, which also already starts with the third-worst result, but then even shows a drastical decrease of 121 % (L_n to M_n) and 857 % (M_n to H_n).

Additionally, ABC is the only algorithm which shows better Silhouette coefficient results for Worker Peer clustering than for Storage Peer clustering.

In figure 6.12 the Averaged Dissimilarity coefficient results for all benchmarked algorithms are depicted. Figure 6.13 provides the comparison of the Averaged Dissimilarity coefficient

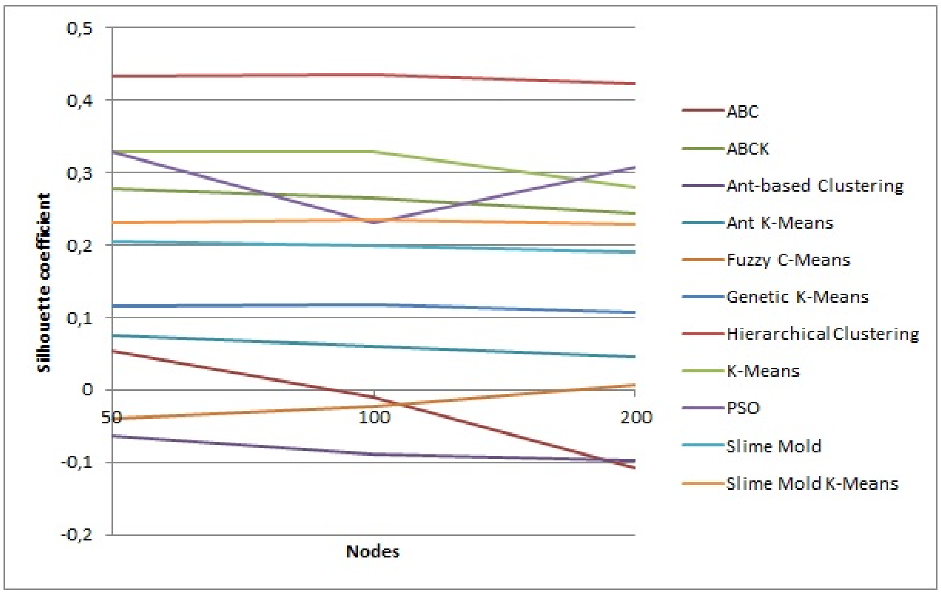

Figure 6.9: Silhouette coefficient results for both Node Peer types and all network size levels.

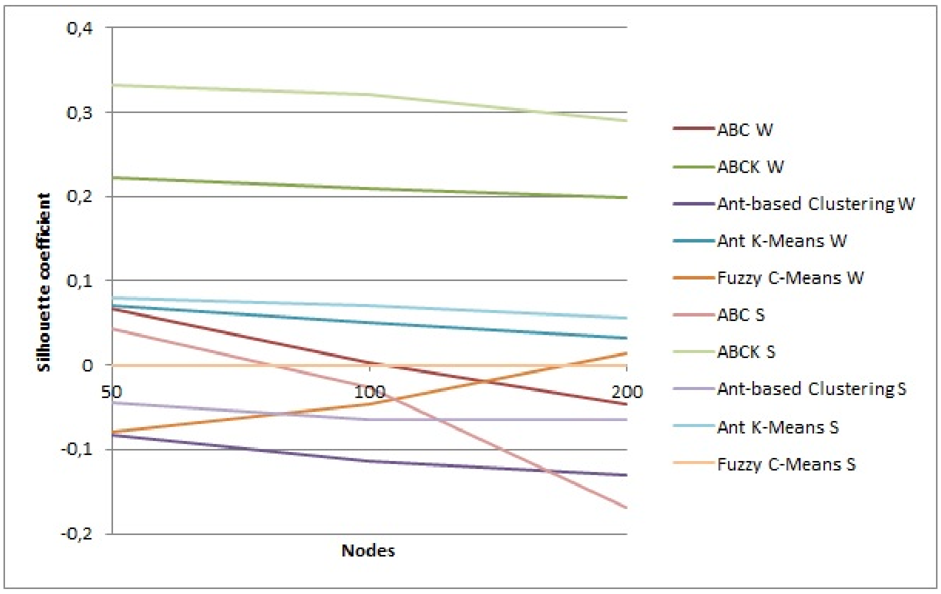

Figure 6.10: Comparison of Silhouette coefficient results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means for all network size levels.

results between Worker Peer clustering and Storage Peer clustering for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means, while figure 6.14 provides it for

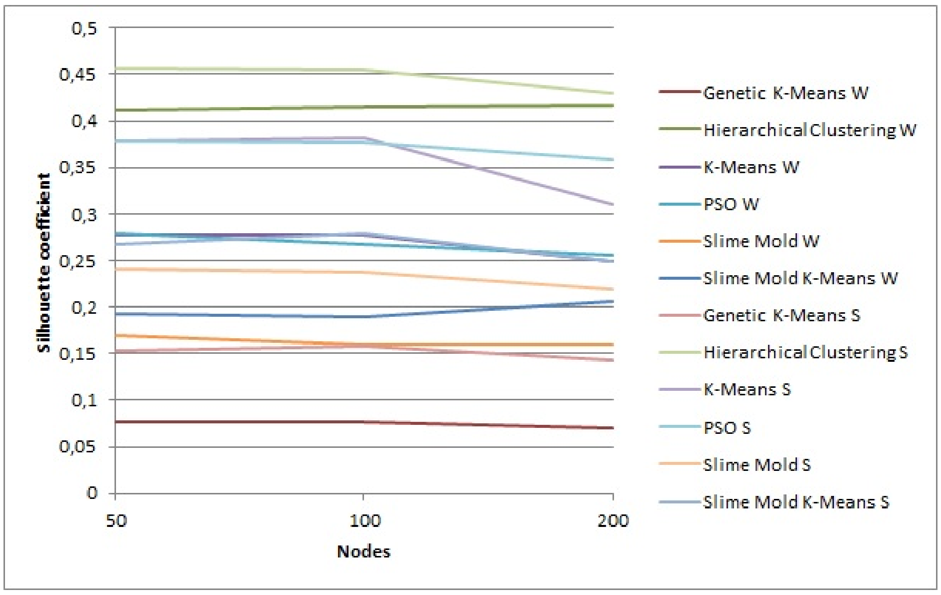

Figure 6.11: Comparison of Silhouette coefficient results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels.

Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means.
As mentioned in section 4.4.5, the smaller the Averaged Dissimilarity coefficient is, the better is the clustering result.

Also for the Average Dissimilarity coefficient Hierarchical Clustering shows the best result, being by 25 % (50 nodes), 28 % (100 nodes) and 24 % (200 nodes) more effective than its successor PSO, which is followed immediately by K-Means with similar results.
Again, Slime Mold and Slime Mold K-Means are placed in the midfield. They provide rather similar results, while both slightly have a increase of 3 % and 1 % for network size level L_n to M_n and then a decrease of 3 % and 2 % for network size level M_n to H_n, respectively.
The Ant-based Clustering algorithm again performs worst regarding the Average Dissimilarity coefficient, just as for the Davies-Bouldin index and the Silhouette coefficient.
While all benchmarked peer clustering algorithms scale rather fairly regarding the Averaged Dissimilarity coefficient, ABC falls out of line and increases massively by 14 % (L_n to M_n) and 36 % (M_n to H_n), which may be caused by the fact that the algorithm does not apply any local optimization such as an additional K-Means step like its modification ABCK.

Furthermore, ABC, just like all other algorithms, first shows better results for Storage Peer clustering than for Worker Peer clustering, but for 200 nodes has better performance when clustering Worker Peers.

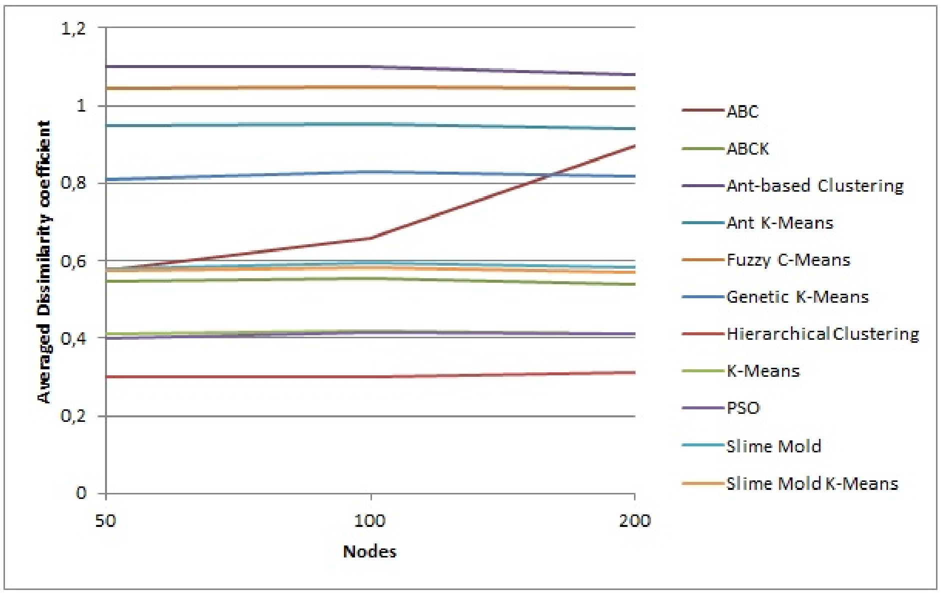

Figure 6.12: Averaged Dissimilarity coefficient results for both Node Peer types and all network size levels.

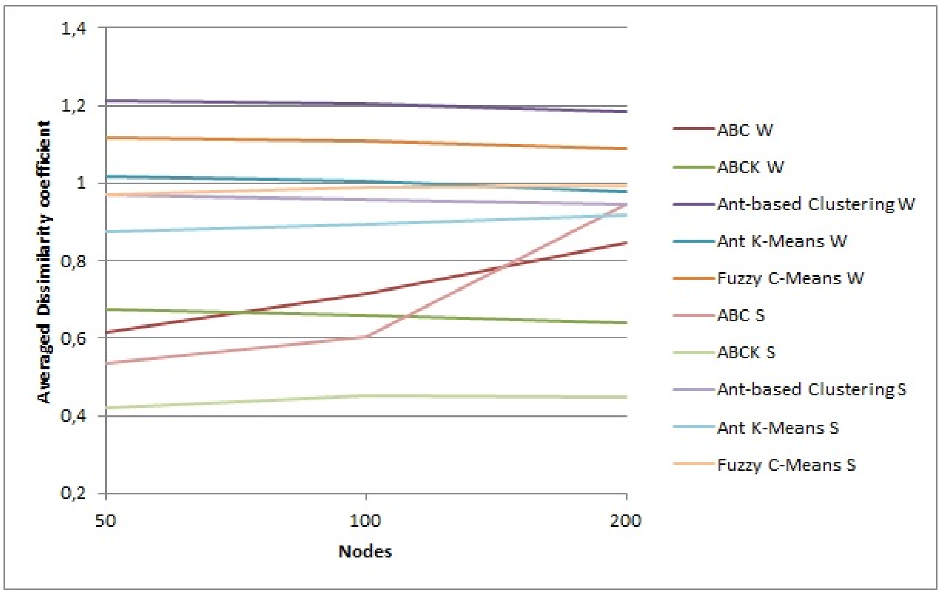

Figure 6.13: Comparison of Averaged Dissimilarity coefficient results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means for all network size levels.

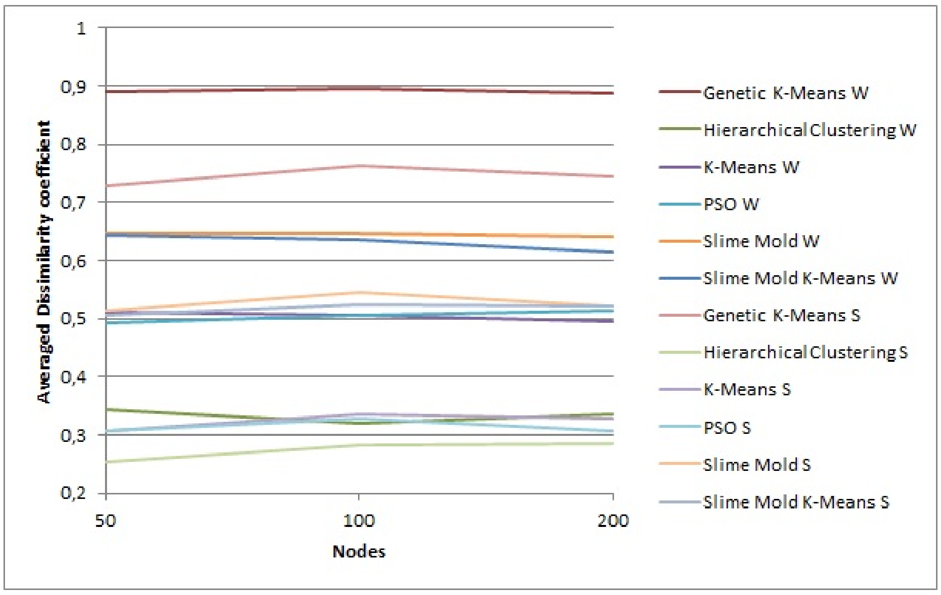

Figure 6.14: Comparison of Averaged Dissimilarity coefficient results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels.

6.5 Statistical Significance of the Results

In order to evaluate and compare Slime Mold and Slime Mold K-Means to the other benchmarked peer clustering algorithms profoundly, a statistical analysis as described in [46] is done.

In detail, one-way ANOVA tests are performed with the following setup [46]:
H_0 is the null-hypothesis, which states that no significant difference between metric M of the two algorithms A and B exists, whereas H_1 is the corresponding alternative-hypothesis. Consequently, if H_0 is rejected, there is not enough data available to determine whether algorithm A or B is significantly better than the other one. Furthermore, H_1 is concluded if H_0 is rejected [46].

Slime Mold and Slime Mold K-Means are tested separately for all combinations of network size levels (L_n, M_n, H_n) and exactly three queries and job requests (cf. section 6.1). As metric M the Averaged Dissimilarity coefficient, explained in detail in section 4.4.5, is chosen, as it was especially developed in the course of this thesis and it particularly focuses on the cohesion within the clusters. The significance level for the ANOVA tests is chosen as $\alpha = 0.05$ [46].

At first, Slime Mold is taking the role of algorithm A, whereas all other benchmarked algorithms, including Slime Mold K-Means, embody algorithm B. Table 6.35 shows the results of these tests.
After that, Slime Mold K-Means embodies algorithm A, while all other benchmarked algorithms, including Slime Mold, take the role of algorithm B. These test results are shown in table 6.36.
For both result tables it applies, if H_0 is concluded, the column h in the table has the value 0. If there is a significant difference between the tested peer clustering algorithms, i.e. H_0 is rejected, h takes the value 1 if algorithm A performs significantly better than algorithm B, otherwise h has the value -1.

Example For a network size of 100 nodes, Slime Molds Averaged Dissimilarity coefficient is 0.5959 with a standard deviation of 0.0002. For the same configuration, ABC has an Averaged Dissimilarity coefficient of 0.6583 with a standard deviation of 0.0003. The result of the one-way ANOVA test is $p = 0.001271288$.
The significance level is $\alpha = 0.05$ and as $\alpha > p$, the null hypothesis H_0 is rejected. Thus, the alternative-hypothesis H_1 is concluded.
The test proves that there exists a significant difference of the Averaged Dissimilarity coefficient metric values. Therefore, Slime Mold performs significantly better than ABC at this configuration and h is set to 1.

As shown in table 6.35, while for a network size of 50 nodes, no significant difference between Slime Mold and ABC and ABCK, respectively, can be determined, Slime

Mold performs significantly better than ABC but is outperformed by ABCK on network levels M_n and H_n. For all other benchmarked algorithms the result is the same for all network size levels: While Slime Mold performs significantly better than Ant-based Clustering, Ant K-Means, Fuzzy C-Means and Genetic K-Means, it is outperformed by Hierarchical Clustering, K-Means and PSO in terms of the Averaged Dissimilarity coefficient.
As can be seen in table 6.36, Slime Mold K-Means achieves rather similar results but exactly the same final outcomes regarding significant differences.
Also on each network size level no significant difference between Slime Mold and Slime Mold K-Means exists.

	mean ± stdev	**p-value**	**h**
	50 nodes		
ABC	0.5755 ± 0.0008	0.790185171	0
ABCK	0.5470 ± 0.0008	0.073269084	0
Ant-based Clustering	1.0986 ± 0.0007	5.77062E-10	1
Ant K-Means	0.9472 ± 0.0001	5.65653E-10	1
Fuzzy C-Means	1.0423 ± 0.0001	5.55717E-11	1
Genetic K-Means	0.8099 ± 0.0003	6.20493E-08	1
Hierarchical Clustering	0.2992 ± 0.0003	1.21269E-08	-1
K-Means	0.4095 ± 0.0005	1.84906E-06	-1
PSO	0.3994 ± 0.0003	4.03503E-07	-1
Slime Mold	0.5798 ± 0.0004	-	-
Slime Mold K-Means	0.5742 ± 0.0002	0.63865944	0
	100 nodes		
ABC	0.6583 ± 0.0003	0.000417622	1
ABCK	0.5540 ± 0.0001	0.001271288	-1
Ant-based Clustering	1.1002 ± 0.00023	1.65439E-11	1
Ant K-Means	0.9531 ± 0.0002	2.38008E-10	1
Fuzzy C-Means	1.0476 ± 2.57904E-05	5.79645E-12	1
Genetic K-Means	0.8289 ± 0.0002	9.53206E-09	1
Hierarchical Clustering	0.3015 ± 0.0001	6.28362E-10	-1
K-Means	0.4210 ± 0.0001	3.0251E-08	-1
PSO	0.4165 ± 0.0003	1.6151E-07	-1
Slime Mold	0.5959 ± 0.0002	-	-
Slime Mold K-Means	0.5810 ± 0.0001	0.130135878	0
	200 nodes		
ABC	0.8957 ± 0.0220	0.001496328	1
ABCK	0.5396 ± 0.0004	0.004216221	-1
Ant-based Clustering	1.0802 ± 0.0003	3.19957E-11	1
Ant K-Means	0.9407 ± 2.90152E-05	1.03927E-11	1
Fuzzy C-Means	1.0420 ± 1.15583E-05	9.97955E-13	1
Genetic K-Means	0.8159 ± 0.0002	2.82205E-09	1
Hierarchical Clustering	0.3108 ± 7.25302E-05	2.23741E-10	-1
K-Means	0.4109 ± 5.80434E-05	6.94393E-09	-1
PSO	0.4099 ± 7.33879E-05	8.5432E-09	-1
Slime Mold	0.5809 ± 0.0002	-	-
Slime Mold K-Means	0.5685 ± 0.0001	0.136412571	0

Table 6.35: Slime Mold ANOVA results.

	mean ± stdev	**p-value**	**h**
	50 nodes		
ABC	0.5755 ± 0.0008	0.927539959	0
ABCK	0.5470 ± 0.0008	0.095272996	0
Ant-based Clustering	1.0986 ± 0.0007	2.14993E-10	1
Ant K-Means	0.9472 ± 0.0001	6.28834E-11	1
Fuzzy C-Means	1.0423 ± 6.00725E-05	4.33966E-12	1
Genetic K-Means	0.8099 ± 0.0003	1.15244E-08	1
Hierarchical Clustering	0.2992 ± 0.0003	3.06358E-09	-1
K-Means	0.4095 ± 0.0005	8.33325E-07	-1
PSO	0.3994 ± 0.0003	1.16171E-07	-1
Slime Mold	0.5798 ± 0.0003	0.63865944	0
Slime Mold K-Means	0.5742 ± 0.0002	-	-
	100 nodes		
ABC	0.6583 ± 0.0003	4.63782E-05	1
ABCK	0.5540 ± 0.0001	0.005729103	-1
Ant-based Clustering	1.1002 ± 0.0002	4.18507E-12	1
Ant K-Means	0.9531 ± 0.0002	5.35059E-11	1
Fuzzy C-Means	1.0476 ± 2.57904E-05	5.68231E-13	1
Genetic K-Means	0.8289 ± 0.0002	2.01672E-09	1
Hierarchical Clustering	0.3015 ± 0.0001	2.38163E-10	-1
K-Means	0.4210 ± 0.0001	1.38983E-08	-1
PSO	0.4165 ± 0.0003	1.36687E-07	-1
Slime Mold	0.5959 ± 0.0002	0.130135878	0
Slime Mold K-Means	0.5810 ± 0.0001	-	-
	200 nodes		
ABC	0.8957 ± 0.0220	0.001170781	1
ABCK	0.5396 ± 0.0004	0.018074686	-1
Ant-based Clustering	1.0802 ± 0.0003	1.49938E-11	1
Ant K-Means	0.9407 ± 2.90152E-05	1.54553E-12	1
Fuzzy C-Means	1.0420 ± 1.15583E-05	1.29002E-13	1
Genetic K-Means	0.8159 ± 0.0002	8.03541E-10	1
Hierarchical Clustering	0.3108 ± 7.25302E-05	8.94089E-11	-1
K-Means	0.4109 ± 5.80434E-05	3.17742E-09	-1
PSO	0.4099 ± 7.33879E-05	4.31318E-09	-1
Slime Mold	0.5809 ± 0.0002	0.136412571	0
Slime Mold K-Means	0.5685 ± 0.0001	-	-

Table 6.36: Slime Mold K-Means ANOVA results.

6.6 Summary

In the beginning of this Chapter the benchmark methodology is described. Thereafter, an extensive Sensitivity Analysis is presented, which serves to find the optimal parameter combination for each benchmarked algorithm. Those parameter values are then used for the competitive benchmarks. Then, the results of the competitive benchmarks are presented, which include eleven peer clustering algorithms, all shortly introduced in section 2.3.

While the Ant-based Clustering algorithm impresses with its very constant execution time, it sticks out with bad quality results compared to the other algorithms.
On the contrary, Slime Mold and Slime Mold K-Means at first glance never belong to the algorithms providing the most effective clustering results, but are always midranged. Nevertheless, they scale very well in terms of effectiveness and time, as they never appear to show unwanted massive increases or decreases in clustering effectiveness results. Therefore, they are likely to provide very satisfying results in larger scaled environments.
As the Hierarchical Clustering algorithm is one of the conventional algorithms, it definitely impresses with short execution times. Furthermore, it also shows to be rather effective, especially when examining the Davies-Bouldin index, the Silhouette coefficient and the Average Dissimilarity coefficient results. Regarding the Dunn index, it was only outperformed by Ant K-Means and Genetic K-Means, which both have shown to be lacking in terms of execution time.

CHAPTER 7

Conclusion and Future Work

In this Chapter possible improvements to the SIPCA framework, which could be made in the future, are discussed. Afterwards, a final conclusion to this thesis is drawn.

7.1 Future Work

After the completion of this thesis, there are still some possible topics for future research work with regard to the SIPCA framework:

- **Combination of different algorithms:** As mentioned in section 4.6, the SIPCA framework would offer the possibility to mix different peer clustering algorithms. Therefore, it would be possible to investigate the combination of algorithms, for example, in order to determine whether two algorithms obtain great results as they are able to complement each other.
  Thus, the combination of different algorithms would offer an interesting field of research.

- **Combination of different approaches:** As briefly addressed in section 1.1, a topic of interest is to investigate the effectiveness of combining peer clustering with load balancing and load clustering, as it would offer even more sophisticated ways of resource allocation.

- **Churn:** In section 6.1 it is mentioned, that no churn (i.e. the fluctuation of node participation, cf. section 2.1) is simulated during the benchmark execution. Nevertheless, as the fact of peers entering and leaving the network dynamically is a major topic in real-world applications, this would offer an interesting field of research, especially with the above mentioned combination of algorithms.

- **Usability & Output Visualization:** Especially regarding the usability of the framework is room for advancement. The development of a graphical user interface would be a huge improvement of the user experience, as it would facilitate the configuration and execution of benchmarks in the framework.
  Furthermore, a great advantage would be the addition of more default output options of the benchmark results, such as the automatic creation of plots. This visualization of the benchmark output would allow the user to immediately obtain an overview of the benchmark results without investing time for visualization.

- **Large Scale Tests:** The benchmark results, presented in Chapter 6, show that multiple algorithms, notably Hierarchical Clustering, Slime Mold and Slime Mold K-Means, perform well in terms of time and quality, even when the amount of network nodes increases.
  An interesting point of research would be the investigation, if this trend is able to continue as the network size keeps increasing. Thus, benchmarks in larger environments would be of great interest.

Regarding the implementation of Slime Mold and Slime Mold K-Means the following aspects could lead to an improvement when being researched:

- **Vegetative Movement:** In the vegetative movement phase amoebae are initialized randomly, i.e. the given number of cluster means are initialized randomly, as it is done for many other algorithms. An interesting approach to analyze would be to investigate the algorithms' behavior and performance when using a more sophisticated way of cluster mean initialization.

- **Combination with Hierarchical Clustering:** Although the Slime Mold combination with K-Means shows fairly satisfactory results, a combination of Slime Mold with Hierarchical Clustering would be an interesting research issue, as the Hierarchical Clustering algorithm thoroughly provides top results in terms of time and effectiveness.

7.2 Conclusion

The main goal of this thesis is the creation of a Peer Model based peer clustering framework. It shall allow the fair and systematic benchmarking and comparison of peer clustering algorithms. Furthermore, the framework shall enable the plugging of different peer clustering algorithms, in such a way that easy exchangeability of the applied algorithms is possible.

The framework is implemented on top of the Peer Model, which allows the frameworks component-based architecture. Additionally, while there are many extensive in-depth configurations offered by the framework for advanced users, for the majority of configuration parameters default values are given. This allows new users to directly focus on

benchmarking.
The algorithms are implemented using generic interfaces. Therefore, the algorithms can be exchanged easily. Also methods for distance measurement and metrics to evaluate the clustering results are implemented in such a Plug & Play manner. Thus, other types of distance measurements and metrics can be added without great effort.

While many well-known conventional and swarm-based clustering algorithms are implemented for benchmarking, also two innovative algorithms imitating the life-cycle of the Dictyostelium discoideum slime mold are implemented, namely Slime Mold and Slime Mold K-Means. They are based on a swarm intelligent algorithm Šešum-Čavić et. al. present in [45].
All implemented algorithms are benchmarked and evaluated using the SIPCA benchmarking framework.

Although Slime Mold and Slime Mold K-Means at first glance never belong to the most effective peer clustering algorithms according to the benchmark results, they scale very well regarding execution time and effectiveness. They are always midranged, but never seem to provide unwanted massive variation in clustering effectiveness results.
On the contrary, many other algorithms either show unwanted increase or decrease, or while impressing with great effectiveness, showing bad results in terms of execution time, or vice versa.
Surprisingly, the Hierarchical Clustering algorithm outperforms all other implemented algorithms. As it is a conventional algorithm, it definitely impresses with short execution times. But it also shows to be rather effective, especially when investigating the results for the Davies-Bouldin index, the Silhouette coefficient and the Averaged Dissimilarity coefficient.

This work shows that it is possible to provide a framework allowing the systematic evaluation and comparison of peer clustering algorithms. Furthermore, two innovative peer clustering algorithms are presented which outperform all other swarm-inspired algorithms.

List of Figures

2.1 Example for a peer, including a sub-peer, a space peer and a wiring. . . . 8

3.1 Local Peer Pattern. 16
3.2 Local Worker Peer Pattern. 16
3.3 Basic Pattern Composition. 17
3.4 Local pattern composition in the framework. 18
3.5 Local inter-node collaboration with UML-like sequence diagram. 19
3.6 Distributed composition of two SIPCA Peer Spaces. 19
3.7 Distributed inter-node collaboration on synchronization and clustering. . 20
3.8 Distributed composition of a Statistics Peer Space and a SIPCA Peer Space. 22
3.9 Distributed inter-node collaboration on statistics. 23
3.10 Wirings of the Coordination Peer, part 1. 26
3.11 Wirings of the Coordination Peer, part 2. 27
3.12 Wirings of the Coordination Peer, part 3. 30
3.13 Wirings of the Coordination Peer, part 4. 33
3.14 Wirings of the Storage Peer. 36
3.15 Wirings of the Worker Peer. 37
3.16 Wiring of the Job Handler Peer. 39
3.17 Wiring of the Client Peer. 40
3.18 Wirings of the Statistics Peer. 44

4.1 Class NodePeer with its sub-classes StoragePeer and WorkerPeer. For reasons of simplicity, getter and setter methods are omitted. 53
4.2 Class EntrySerialization. For reasons of simplicity, getter and setter methods are omitted. 54
4.3 Interface IAlgorithm. 54
4.4 Interface IDistance. 54
4.5 Interface IEvaluation. 55

5.1 Life cycle of the Dictyostelium discoideum slime mold [20]. 78

6.1 Execution time results for both Node Peer types and all network size levels. 101
6.2 Execution time results zoomed in. 101
6.3 Davies-Bouldin results for both Node Peer types and all network size levels. 103

6.4 Comparison of Davies-Bouldin index results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering and Ant K-Means for all network size levels. 103
6.5 Comparison of Davies-Bouldin index results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels. 104
6.6 Dunn index results for both Node Peer types and all network size levels. . 105
6.7 Comparison of Dunn index results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means for all network size levels. 105
6.8 Comparison of Dunn index results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels. 106
6.9 Silhouette coefficient results for both Node Peer types and all network size levels. 107
6.10 Comparison of Silhouette coefficient results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means for all network size levels. 107
6.11 Comparison of Silhouette coefficient results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels. 108
6.12 Averaged Dissimilarity coefficient results for both Node Peer types and all network size levels. 109
6.13 Comparison of Averaged Dissimilarity coefficient results between Worker and Storage Peers for ABC, ABCK, Ant-based Clustering, Ant K-Means and Fuzzy C-Means for all network size levels. 109
6.14 Comparison of Averaged Dissimilarity coefficient results between Worker and Storage Peers for Genetic K-Means, Hierarchical Clustering, K-Means, PSO, Slime Mold and Slime Mold K-Means for all network size levels. 110

List of Tables

4.1 Benchmark metrics. 58
4.2 SIPCA Peer Space parameters, part 1. 59
4.3 SIPCA Peer Space parameters, part 2. 60

5.1 Basic characteristics of ABC. 71
5.2 Basic characteristics of ABCK. 72
5.3 Basic characteristics of Ant-based Clustering. 74
5.4 Basic characteristics of Ant K-Means. 75
5.5 Basic characteristics of PSO. 77
5.6 Basic characteristics of Slime Mold. 80
5.7 Basic characteristics of Slime Mold K-Means. 81

6.1 ABC parameter values before Sensitivity Analysis. 86
6.2 ABC Sensitivity Analysis results. 86
6.3 ABCK parameter values before Sensitivity Analysis. 86
6.4 ABCK Sensitivity Analysis results. 87
6.5 Ant-based Clustering parameter values before Sensitivity Analysis. 87
6.6 Ant-based Clustering Sensitivity Analysis results. 87
6.7 Ant K-Means parameter values before Sensitivity Analysis. 88
6.8 Ant K-Means Sensitivity Analysis results. 88
6.9 Fuzzy C-Means parameter values before Sensitivity Analysis. 89
6.10 Fuzzy C-Means Sensitivity Analysis results. 89
6.11 Genetic K-Means parameter values before Sensitivity Analysis. 89
6.12 Genetic K-Means Sensitivity Analysis results. 90
6.13 Hierarchical Clustering parameter values before Sensitivity Analysis. . . . 90
6.14 Hierarchical Clustering Sensitivity Analysis results. 90
6.15 K-Means parameter values before Sensitivity Analysis. 91
6.16 K-Means Sensitivity Analysis results. 91
6.17 PSO parameter values before Sensitivity Analysis. 91
6.18 PSO Sensitivity Analysis results. 92
6.19 Slime Mold parameter values before Sensitivity Analysis. 92
6.20 Slime Mold Sensitivity Analysis results. 93
6.21 Slime Mold K-Means parameter values before Sensitivity Analysis. 93

6.22 Slime Mold K-Means Sensitivity Analysis results. 93
6.23 Metric abbreviation explained for competitive benchmark. 94
6.24 ABC raw result data. 94
6.25 ABCK raw result data. 95
6.26 Ant-based Clustering raw result data. 95
6.27 Ant K-Means raw result data. 96
6.28 Fuzzy C-Means raw result data. 96
6.29 Genetic K-Means raw result data. 97
6.30 Hierarchical Clustering raw result data. 97
6.31 K-Means raw result data. 98
6.32 PSO raw result data. 98
6.33 Slime Mold raw result data. 99
6.34 Slime Mold K-Means raw result data. 99
6.35 Slime Mold ANOVA results. 113
6.36 Slime Mold K-Means ANOVA results. 114

List of Algorithms

5.1 Artificial Bee Colony algorithm 69
5.2 Artificial Bee Colony algorithm combined with K-Means 72
5.3 Algorithm for ant-based clustering 73
5.4 Algorithm for Ant K-Means 75
5.5 Algorithm for Particle Swarm Optimization 76
5.6 Algorithm for Slime Mold 79
5.7 Algorithm for Slime Mold K-Means 81

Acronyms

ABC	Artificial Bee Colony.
ABCK	Artificial Bee Colony combined with K-Means.
ADC	Averaged Dissimilarity coefficient.
API	Application Programming Interface.
CPU	Central Processing Unit.
DBI	Davies-Bouldin index.
DI	Dunn index.
P2P	Peer-to-Peer.
PIC	peer-in-container.
POC	peer-out-container.
PSO	Particle Swarm Optimization.
RAM	Random Access Memory.
SC	Silhouette Coefficient.
SIPCA	Self-Initiative Peer Clustering Agents.
SON	Semantic Overlay Network.
TTL	time-to-live.
TTS	time-to-start.
TWCV	Total within cluster variation.
URI	Unique Resource Identifier.

Bibliography

[1] S. Androutsellis-Theotokis and D. Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, Dec. 2004.

[2] G. Armano and M. Farmani. Clustering analysis with combination of artificial bee colony algorithm and k-means technique. *International Journal of Computer Theory and Engineering*, 6:141–145, 01 2014.

[3] J. C. Bezdek. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Kluwer Academic Publishers, Norwell, MA, USA, 1981.

[4] J. C. Bezdek, R. Ehrlich, and W. Full. FCM—the Fuzzy C-Means clustering-algorithm. *Computers & Geosciences*, 10:191–203, 12 1984.

[5] U. Boryczka. Ant clustering algorithm. *Intelligent Information Systems*, 1998, 01 2008.

[6] J. F. Buford and H. Yu. Peer-to-peer networking and applications: Synopsis and research directions. In *Handbook of Peer-to-Peer Networking*, pages 3–45, 10 2010.

[7] G. Cao, D. Song, and P. Bruza. Fuzzy k-means clustering on a high dimensional semantic space. In *Advanced Web Technologies and Applications*, pages 907–911, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[8] S. Cejka. Enabling scalable collaboration by introducing platform-independent communication for the peer model. Master's thesis, Vienna Univerity of Technology, 2019.

[9] S. Craß, J. Hirsch, E. Kühn, and V. Sesum-Cavic. Modeling a flexible replication framework for space-based computing. In *Software Technologies*, pages 256–272, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[10] A. Crespo and H. Garcia-Molina. Semantic overlay networks for p2p systems. In *Agents and Peer-to-Peer Computing*, pages 1–13, Berlin, Heidelberg, 2005. Springer.

[11] D. L. Davies and D. W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-1(2):224–227, 1979.

[12] C. H. Ding, S. Nutanong, and R. Buyya. Peer-to-peer networks for content sharing. In *Peer-to-Peer Computing: The Evolution of a Disruptive Technology*, pages 28–65, Hershey, PA, 01 2004. IGI Global.

[13] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[14] J. C. Dunn. A fuzzy relative of the isodata process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, 3(3):32–57, 1973.

[15] J. C. Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of Cybernetics*, 4(1):95–104, 1974.

[16] K. C. Gowda and G. Krishna. Agglomerative clustering using the concept of mutual nearest neighbourhood. *Pattern Recognition*, 10:105–112, 1978.

[17] K. C. Gowda and T. Ravi. Divisive clustering of symbolic objects using the concepts of both similarity and dissimilarity. *Pattern Recognition*, 28:1277–1282, 1995.

[18] J. Handl and B. Meyer. Improved ant-based clustering and sorting in a document retrieval interface. In *Parallel Problem Solving from Nature — PPSN VII*, pages 913–923, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[19] J. Handl and B. Meyer. Ant-based and swarm-based clustering. *Swarm Intelligence*, 1(2):95–113, Dec 2007.

[20] B. Hollis. Rapid antagonistic coevolution between strains of the social amoeba dictyostelium discoideum. *Proceedings of the Royal Society B: Biological Sciences*, 279(1742):3565–3571, 2012.

[21] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, Sept. 1999.

[22] D. Karaboga and C. Ozturk. A novel clustering approach: Artificial bee colony (abc) algorithm. *Applied Soft Computing*, 11(1):652 – 657, 2011.

[23] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., 1990.

[24] R. H. Kessin. *Dictyostelium: Evolution, Cell Biology, and the Development of Multicellularity*, volume 38 of *Developmental and Cell Biology Series*. Cambridge University Press, 2001.

[25] M. Khambatti, K. D. Ryu, and P. Dasgupta. Structuring peer-to-peer networks using interest-based communities. In *Databases, Information Systems, and Peer-to-Peer Computing*, pages 48–63, Berlin, Heidelberg, 2004. Springer.

[26] K. Krishna and M. Murty. Genetic k-means algorithm. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 29:433–9, 02 1999.

[27] E. Kühn, S. Craß, G. Joskowicz, A. Marek, and T. Scheller. Peer-based programming model for coordination patterns. In *Coordination Models and Languages*, pages 121–135, Berlin, Heidelberg, 2013. Springer.

[28] e. Kühn, S. Craß, G. Joskowicz, and M. Novak. Flexible modeling of policy-driven upstream notification strategies. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, SAC '14, pages 1352–1354, New York, NY, USA, 2014. ACM.

[29] E. Kühn, A. Marek, T. Scheller, V. Sesum-Cavic, M. Vögler, and S. Craß. A space-based generic pattern for self-initiative load clustering agents. In *Coordination Models and Languages*, pages 230–244, Berlin, Heidelberg, 2012. Springer.

[30] R. Kuo, H. Wang, T.-L. Hu, and S. Chou. Application of ant k-means on clustering analysis. *Computers & Mathematics with Applications*, 50(10):1709 – 1724, 2005.

[31] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710, Feb 1966.

[32] B. Liang, J. Tang, J. Li, and K. Wang. Keyword extraction based peer clustering. In *Grid and Cooperative Computing - GCC 2004*, pages 827–830, Berlin, Heidelberg, 2004. Springer.

[33] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005.

[34] E. D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *Proceedings of the Third International Conference on Simulation of Adaptive Behavior : From Animals to Animats 3: From Animals to Animats 3*, SAB94, pages 501–508, Cambridge, MA, USA, 1994. MIT Press.

[35] A. Modarresi, A. Mamat, H. Ibrahim, and N. Mustapha. A social network peer-to-peer model for peer clustering. In *2008 International Symposium on Information Technology*, volume 3, pages 1–7, Aug 2008.

[36] D. R. Monismith. *The Uses of the Slime Mold Lifecycle as a Model for Numerical Optimization*. PhD thesis, Oklahoma State University, USA, 2010.

[37] M. Omran, A. Salman, and A. Engelbrecht. Image classification using particle swarm optimization. *Simulated Evolution and Learning*, 1:370–374, 08 2004.

[38] R. Parpinelli and H. Lopes. New inspirations in swarm intelligence: A survey. *IJBIC*, 3:1–16, 01 2011.

[39] P. J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53 – 65, 1987.

[40] V. Šešum-Čavić and E. Kühn. Chapter 8 self-organized load balancing through swarm intelligence. In *Next Generation Data Technologies for Collective Computational Intelligence*, pages 195–224, Berlin, Heidelberg, 2011. Springer.

[41] A. Singh and M. Haahr. Decentralized clustering in pure p2p overlay networks using schelling's model. In *2007 IEEE International Conference on Communications*, pages 1860–1866, June 2007.

[42] P.-N. Tan, M. Steinbach, and V. Kumar. 2. data. In *Introduction to Data Mining, (First Edition)*, chapter 2. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[43] P.-N. Tan, M. Steinbach, and V. Kumar. 8. cluster analysis: Basic concepts and algorithms. In *Introduction to Data Mining, (First Edition)*, chapter 8. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[44] S. Zischka. A coordination-based framework for routing algorithms in unstructured peer-to-peer networks. Master's thesis, Vienna Univerity of Technology, 2017.

[45] V. Šešum Čavić, E. Kühn, and D. Kanev. Bio-inspired search algorithms for unstructured p2p overlay networks. *Swarm and Evolutionary Computation*, 29:73 – 93, 2016.

[46] V. Šešum Čavić, E. Kühn, and D. Kanev. Bio-inspired search algorithms for unstructured p2p overlay networks. *Swarm and Evolutionary Computation*, 29:73 – 93, 2016.

Web-References

[47] Google Compute Engine. https://cloud.google.com/compute/. Accessed: 2020-07-19.

[48] Google Compute Engine CPU platforms. https://cloud.google.com/compute/docs/cpu-platforms. Accessed: 2020-07-19.

[49] Java Standard Edition 8. https://docs.oracle.com/javase/8/. Accessed: 2019-06-29.

[50] Percentage Difference. https://www.mathsisfun.com/percentage-difference.html. Accessed: 2020-12-06.