Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diploma thesis Nr. 3636

Scalable data retrieval in a mobile environment

Christian Mayer

Course of Study: Information Science

Examiner: Prof. Dr. rer. nat. Kurt Rothermel

Supervisor: Dr. rer. nat. Muhammad Adnan Tariq

Commenced: January 13, 2014

Completed: July 15, 2014

CR-Classification: C.2.4, H.3.3, H.3.4

Abstract

Retrieving multidimensional data out of distributed systems becomes increasingly important. But applications of these systems are often not only interested in data vectors that match certain queries. Instead, many applications demand for retrieval of data with high quality. In this thesis, we design a distributed system that can be used by applications to retrieve data of high quality for arbitrary multidimensional queries.

Major challenges for the quality-based data retrieval are to 1.) find an appropriate formalization of data quality, 2.) design routing algorithms for queries, that are robust in the presence of high dynamics with respect to the participants of the system and the data on the participants and 3.) handle heterogeneous and high-dimensional data in the system.

In order to retrieve data quality, we propose 1.) the measure of confidence for a query that is based on clusters of data. When a participant of the system finds, that its confidence for a query is high, it will assume to possess data of high quality for that query. 2.) Further, we design and implement routing strategies in order to route queries to nodes that can answer them with high confidence. Maintaining exact routing tables for each possible query would be infeasible, so nodes have to model the data that can be reached via neighbours in routing models. Such modelling of data is based on structural properties of the data such as how good the data can be clustered. 3.) In the high-dimensional space, we have to overcome the curse of dimensionality: the structure of data can become invisible in higher dimensions. We address this problem with a method for dimensionality reduction that reduces the dimensions with highest data variance.

The evaluation of our approaches shows a high accuracy of query routing, even if our approaches do not make use of scalability bottlenecks like flooding of the query or flooding of routing information. Further, we show that the use of dimensionality reduction in routing has positive influence on the routing accuracy. We think that the methods in our approach can be useful instruments, whenever the task of retrieving data of high quality has to be outsourced to a distributed system.

ACKNOWLEDGEMENTS

I would like to express my special appreciation and gratitude to my supervisors Dr. rer. nat. Muhammad Adnan Tariq, who gave me invaluable advice, support and motivation for this thesis, and Dipl.-Inf. Thomas Bach, whose feedback and change of perspective often lead to new ideas. Working together with both demonstrated me the meaning of synergy. *The combined effect of individuals in collaboration [...] exceeds the sum of their individual effects.* - Stephen R. Covey.
I am particularly grateful for the assistance given by Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel, who gave me the possibility to write the thesis at the department of distributed systems and opened new opportunities for my personal advancement. My special thanks to the staff of the department of distributed systems for the gentle and fast help in many situations during my work on this thesis.

I would like to thank my parents Vera and Ulrich Mayer for the inestimable support during my studies. Also, I wish to acknowledge the help provided by my family and friends.

Finally, I would like to express my particular thank to the woman in my life Anna Altimira, who gave me more than I can express here. Last, but not least I would like to thank my little daughter Amalie for the great joy, she brought to my life.

Contents

List of Figures

List of Algorithms

List of Symbols

c	A Cluster of vectors.
$\bar{c}$	The centroid of a cluster c.
$C(c, \mathbf{q})$	Confidence score for query $\mathbf{q}$ and cluster c.
Σ	The covariance matrix of a data set.
V	The set of local data.
DM_i	Data model of node p_i.
$\mathbf{v}$	A single local data vector.
m	Dimensionality of the vector space.
$Dist(c, \mathbf{q})$	Distance score from query $\mathbf{q}$ to cluster c.
$C_{est}(\mathbf{q}, c)$	Estimated confidence score for query $\mathbf{q}$ and cluster c.
h	Hop count/depth of exploration query.
$C_i(\mathbf{q})$	Local confidence of node p_i for query $\mathbf{q}$.
w	Maximal number of query-confidence pairs in a routing model.
μ	Mean value.
$C^{\mathbf{q}}_{min}$	Minimal confidence value for query $\mathbf{q}$.
t	The number of data.
n	The number of nodes in a network.
p	A node in the network.
$Q(c)$	Quality score for a cluster c.
$\mathbf{q}$	Query vector.
RM	Routing model.
s	Selectivity of an exploration query.
σ	Standard deviation.

1 Introduction

The world's data volume grows rapidly. Examples of data are images and videos, telephone calls, emails, articles, web pages and sensor data [BA03]. Because of the high velocity of data generation, it becomes increasingly important to automatically extract relevant information from the data in real-time [GR12]. In addition to data volume and velocity, a major task today is analyzing data with heterogeneous and diverse dimensionality [WZWD14]. Diverse dimensionality can arise, when different organizations or data sources observe individuals with respect to different attributes. When data is of diverse dimensionality, information extraction becomes more difficult. This challenge of high data volume, high velocity of data generation and high dimensionality of data is commonly identified with the term Big Data [Jac09]. Numerous applications and design principles have evolved that are positioned around the field of Big Data. Examples are search engines [BKL08], social networks [Loh12], Internet encyclopedia [CSC+06], cloud computing [ADEA10], pervasive computing [Han03], the Internet of Things [GBMP13] and others.

An application of large amounts of data with heterogeneous dimensionality is given in collective adaptive systems. Collective adaptive systems (CAS) are large-scale distributed systems, that consist of a heterogeneous and dynamic changing set of autonomous participants [ABGS+14]. An example of a CAS is an urban mobility system, where different participants build a spontaneous interaction network. Participants can be agents in the real world like trains, passengers and buses, as well as back-end computer systems that are responsible for payment or route planning. In order to react to changing contexts, the system should be able to store and retrieve observations from past experiences (collaborative learning). The observations can be the basis for diverse learning algorithms. For example, the CAS might have learned that on major events, the number of buses should increase. If the system has made many observations in a similar context, it can be more *confident* about the learning. On the other hand, the system should not have much confidence in the learning, if there are only few observations. This knowledge can be maintained in a distributed knowledge base that consists of local data models on each participant. In addition to storing past observations, data models provide an interpretation of how much confidence in the data a learning algorithm could have. Participants may find different aspects of the environment important. This results in local data models with heterogeneous dimensionality that can change over time. In this context, scalable retrieval of information is a very challenging task.

Such systems are different to traditional data retrieval systems. The aim is not to store and retrieve some specific data. Instead, automatically interpretation of the data is needed, so that information about the data can be retrieved. In the CAS example, data is interpreted in order to retrieve a *confidence score* for certain contexts. This score can then be used for higher levels of learning.

> Here's the big truth about big data in traditional databases: It's easier to get the data in than out.
> - Adam Jacobs [Jac09]

We will concentrate on methods that help to *get something out* of the data in a distributed system. In a nutshell, this thesis is about retrieving specific quality information out of unstructured, multidimensional data that is stored in a distributed system.

1.1 Contributions

The main contributions of this thesis are the following:

- We propose the measure of data **confidence**. With this measure, nodes can find out about the quality of the local data for a given multidimensional query. A higher confidence score means that a node can provide better knowledge for the query. The score depends on the euclidean distance between query and data, the number and density of the data in proximity to the query and the shape of the data.
- We develop a routing approach, so that a query can be routed to a node with high local confidence. Each participant maintains information about the data that is reachable via each neighbour. This information can be used to decide to which neighbour the query should be routed. To spread the routing information across the network, each node aggregates routing information of the neighbours and its local data and forwards the aggregate to the neighbours. Because of this method, we call this approach **data aggregation** (DA) routing approach. While this kind of routing shows a high accuracy, this technique can be inflexible in large and dynamic networks.
- In order to provide scalability, we develop a more fine grained routing approach, where nodes do not have to wait until each reachable participant has forwarded routing information. In the **query learning** (QL) routing approach, nodes use past queries to neighbours to find out about the kind of data that is reachable via this neighbour. Each neighbour feedbacks the queries with an estimation of the confidence score that will be retrieved, if the query is send via itself. Feedback receivers use this information for future routing decisions. The confidence estimation of the neighbours is based on their local data and their local routing information. Because of that, the learning of routing information is only local and the system can adapt itself very dynamically.

- As queries and data can be very high dimensional, we have to take into account the curse of dimensionality [VF05]: with increasing dimensionality, data becomes sparse and clusters become invisible. Therefore, we introduce a **dimensionality reduction** method for the QL routing approach. This enables nodes to ignore dimensions that do not have good clustering properties.
- We investigate the routing approaches in an **evaluation** of the routing accuracy of the different approaches with and without dimensionality reduction.

1.2 Structure of the Thesis

In the following, we give the structure of the thesis. Chapter 2 is about related work on which this thesis is built-on. The assumptions about the network and the problem formulation is presented in Chapter 3. Afterwards, we introduce the confidence metric and how we came to it in Chapter 4. In Chapter 5, we present the DA routing approach that is based on forwarding cluster information. Nodes use this information to route queries through the network. As this approach is not always feasible in large and dynamic networks, we develop the QL routing approach in Chapter 6. Here, the learning of routing information is based on exploration queries and their feedbacks. For the QL routing approach, we give a method for reducing high-dimensional data. Both approaches will be evaluated in Chapter 7. We summarize the results of the thesis in Chapter 8 and propose topics for future work on the approaches.

2 Related Work

In this chapter, we give an overview about related approaches for routing and data retrieval in dynamic networks. Initially, we investigate diverse strategies for data retrieval in a peer-to-peer system. Afterwards, we introduce publish/subscribe systems and discuss how those systems can be used to retrieve information. Finally, we introduce the method of reinforcement learning that has influenced our routing approach.

2.1 Data Retrieval in Peer-to-peer Systems

Scalable retrieval of data in peer-to-peer (P2P) systems was subject to intensive research during the last decades [LCP+05]. P2P systems are decentralized, distributed systems, where each node can be client and server of the system at the same time. Because of the absence of a centralized component, there is no single point of failure.

Typically P2P systems can be categorized into structured and unstructured systems.

2.1.1 Unstructured P2P Approaches

Unstructured P2P systems do not maintain a fixed structure of the topology and the kind of data that is stored by peers. Each peer can store arbitrary data. Because of this, unstructured P2P systems normally work well, even if there is a high frequency of nodes joining and leaving the system.

When processing queries for data items, unstructured P2P systems often use *blind searches* such as flooding of the query [BAH+06]. In blind searches, peers maintain no routing information about data that is reachable via a neighbour. Examples of such P2P systems using flooding are Gnutella [AH00] and Kazaa [LKR04]. Flooding the query has the property of guaranteed retrieval of the searched file, but limits the scalability of the system. The other extreme in blind searching is the random walk approach, where the query is forwarded to one random neighbour. This solves the problem of high message overhead. Eventually, each content can be found, but the search latency can become very high.

In order to provide scalability, nodes can maintain some routing information to enable an *informed search* [BAH+06]. An example for informed search is given in Freenet [CSWH01], where files are associated with keys and nodes store keys of files that can be retrieved via neighbours. A query (key) can be forwarded to the neighbour that has already answered similar keys.

2.1.2 Structured P2P Approaches

In structured P2P systems, the network overlay is organized in a way, so that contents can be retrieved efficiently. Structured P2P systems can be seen as Distributed Hash Tables (DHTs), because these systems provide key-value lookup like a hash table. The structure of the network helps nodes to route queries efficiently to their neighbours, because a node has information about the responsibility of its neighbours for certain keys.

An example of a structured P2P approaches is CAN [RFH$^+$01], where a key is a point in the multidimensional space. Each node is responsible for keys in a certain area of this multidimensional space. A node is connected with its direct two neighbours in each dimension. When a node receives a query, it determines, whether the key falls in its responsibility area. If this is the case, it will answer it locally, otherwise it forwards the query to the neighbour that is closest to the query. Other examples of structured P2P approaches are Chord [SMLN$^+$03], where the overlay is organized as ring with *long range contacts* to nodes that are far away in the ring, so that the search is more efficient, and Pastry [RD01].

All structured P2P approaches have in common, that data retrieval is efficient, but the maintenance of the structure is costly, when nodes join and leave the protocol very frequently. DHTs are designed to retrieve values, having the associated keys. But as we have already discussed in the introduction, we do not search for exact query matches in this thesis, but for nodes that contain similar data. We investigate now an approach that provides more complex, content based query searches using a hybrid overlay structure.

2.1.3 Semantic Small World

The problem of routing queries to nodes that contain similar data is addressed by the semantic small world protocol [LLS04]. For this, a small world network is constructed. The characteristic properties of such a network are a small average length of query routes and a high clustering coefficient. This means that two neighbours of a node are likely to be already neighbours of each other. We will name such a cluster of nodes *node cluster* to distinguish it clearly from the concept of *data cluster* that is used often in this thesis.

The term *semantic* in semantic small world refers to the idea of building clusters of nodes that store data vectors that are approximately in the same area of the space. For this, each node determines its *semantic label* that is the centroid of the largest cluster of local data. Based on the semantic labels, nodes construct an overlay network, such that semantically close nodes are also topographically close. We can see a simplified example of such a network in Figure 2.1 (please see the paper for a more detailed example). Each node has a semantic label in the m-dimensional space. Additionally, we have plotted the local data vectors of node P_3 that are responsible for its semantic label. Each node maintains information about its current node cluster that is given by the box around a node. Each node maintains communication links i) to each node in its node cluster, ii) to $2m$ nodes of the neighbour node clusters and iii) to some distant nodes (long range contacts) with probability proportional to the inverse distance to that nodes. The links of node P_3 are given in the example.

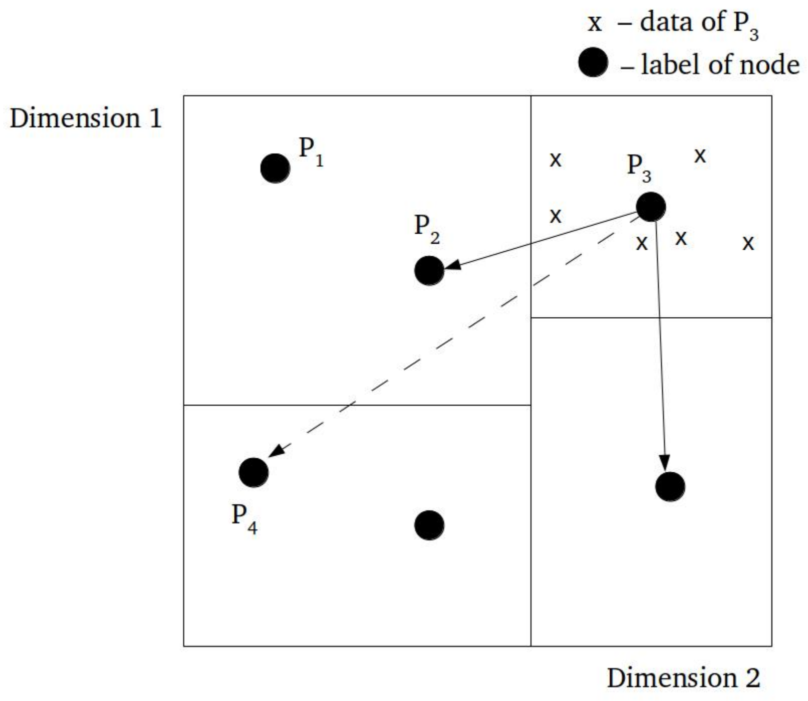


Figure 2.1: An example for semantic small world.

With increasing dimensionality, the number of neighbours of a node increases too. Because of that, the authors give a method to reduce dimensionality of the overlay by constructing a double linked list that connects a node with only two of the nodes in neighbour clusters. Routing is performed by simply forwarding the query to the node whose label is closest to the query.

This approach addresses some of the problems that have to be solved in our approach such as semantic query forwarding and high dimensionality. However, the dimensionality reduction reduces only the overlay, which results in a lower number of links to neighbours. The authors specify explicitly that for the routing, the whole dimensional query is used. Additionally, each node determines its semantic label by clustering its local data vectors in the whole space. This is problematic, because the curse of dimensionality remains a problem as in high dimensions clusters can become invisible.

Another limitation is the assumption that each node maintains data of the same dimensionality. In our approach, we do not assume such homogeneous dimensionality: nodes may have different dimensions.

Because the semantic small world approach is a hybrid between structured and unstructured P2P systems, it still requires a certain decree of overlay maintenance costs. While this design decision enables efficient retrieval of data, it may be better to not assume such a structure in a highly dynamic environment. Using such an unstructured topology will result in suboptimal routing but low maintenance costs of the overlay.

2.2 Publish/Subscribe Systems

In this section, we provide an overview about different types of routing in Publish/Subscribe (pub/sub) systems [BQV05, KDT13, TKKR09, TKK$^+$11, TKKR12, TKR13, TKR14]. A decentralized pub/sub system can be implemented as a P2P system, where communication is performed via events. An event is a set of attribute-value-pairs. Nodes can publish events or subscribe to events. This results in a loose coupling between publisher and subscriber. They do not need to know about each other, because an event is not sent to a specific node. Instead, the system delivers events to subscribers. A subscriber has to specify an event filter that keeps only events, it is interested in. Pub/sub systems can be used to deliver queries (events) to content providers, that subscribe to queries with their content.

There are multiple *subscription models* how nodes can specify their interest in events:

- Using the **topic-based model**, a subscriber receives only events of a certain topic. Normally, topics are very coarse granular. A subscriber can not specify its individual filter and has to rely on the predetermined topics.
- In the **content-based model**, subscribers can specify conditions on the attributes of events, they are interested in.
- When nodes do not know the syntactic and semantic structure of an event, i.e., the meaning, number and type of the attributes is unknown, this information can be encoded in metadata of the events. **Concept-based addressing** makes use of metadata, so that the filtering on the event can be performed on concepts rather than attribute values.

Beside the different subscription models, pub/sub systems differ in the used *event routing algorithms*. In general, one can categorize these algorithms into four different approaches [BQV05]:

Event flooding Publishers broadcast events to each node in the system. Then, nodes will decide locally, if the event matches a subscription of the node. This obviously results in a huge event message overhead.

Subscription flooding Subscribers flood their subscriptions across the network. Hence, each publisher knows locally to which subscriber it has to send the event. If the rate of subscription change is high, this solution will be infeasible due to the overhead of subscription messages.

Filtering-based routing In contrast to the flooding approaches, an event is routed from the publisher to the interested subscribers. Each node stores routing information for each neighbour about the subscriptions that can be reached via the neighbour. A node can aggregate multiple subscriptions, so that it has to forward only aggregated subscriptions. Events were only forwarded to neighbours that have subscribed to those.

Rendezvous-based routing For each event ev and each subscription sn, a set of rendezvous nodes can be computed. A publisher publishes ev by forwarding it to the rendezvous nodes $RV(ev)$. The same will be done, when a node intends to subscribe with sn: it determines a set of rendezvous nodes $RV(sn)$ to which sn is forwarded. If $RV(ev) \cap RV(sn)$ and sn matches ev, then at least one of the nodes in $RV(ev)$ knows a subscriber to ev.

In general, pub/sub systems can be used for routing queries (events) to content providers (subscriber to queries). But as we are interested in scalable protocols, we can not use one of the flooding based event routing approaches. The rendezvous-based approach as presented is very general, finding a rendezvous function is a challenging problem. When nodes come and go very frequently, how to integrate them into the rendezvous functions, so that they participate in the routing?

In our data aggregation (DA) routing approach (see Chapter 5), we have used an idea that is similar to *filtering-based routing*. A node aggregates routing knowledge and forwards the aggregate to neighbours. However, we do not rely on the pub/sub paradigm as there is no hard subscription to queries in our approach. Some nodes may answer a query better than others. To increase robustness, each node should be allowed to answer each query. In the pub/sub paradigm, this would result in a situation, where each node subscribes to each query.

2.3 Collaborative Reinforcement Learning

Collaborative Reinforcement Learning can be used as a method to optimize routing paths in a mobile ad hoc network [BQV05, KV06, BL94, HF10, WHH96].

Reinforcement learning is a technique for a single agent to learn optimal behaviour in an observable environment, where the agent can determine the state of the environment. The agent performs actions (state transitions) to change the state of the environment. Each state transition is rewarded by a *reward function*. The goal of the agent is to maximize the total reward it will receive, when it follows the transition rules.

In collaborative reinforcement learning, the system is assumed to consist of a set of agents, where each agent interacts only with its local environment and follows relative simple rules. In contrast to reinforcement learning, there are multiple agents and each agent can observe only its local reward, but not rewards of other agents [KV06]. However, the goal of each agent is still to maximize the global reward of the system.

The states and state transitions of the collaborative agent system are illustrated in Figure 2.2. There are two agents A and B with internal states A1,A2,A3,B1,B2 and internal and external state transitions. For example, the internal state A3 has two possible state transitions: one to the internal state A2 and another to the external state B1. Agent A has to decide which transition gives more expected reward in the long run.

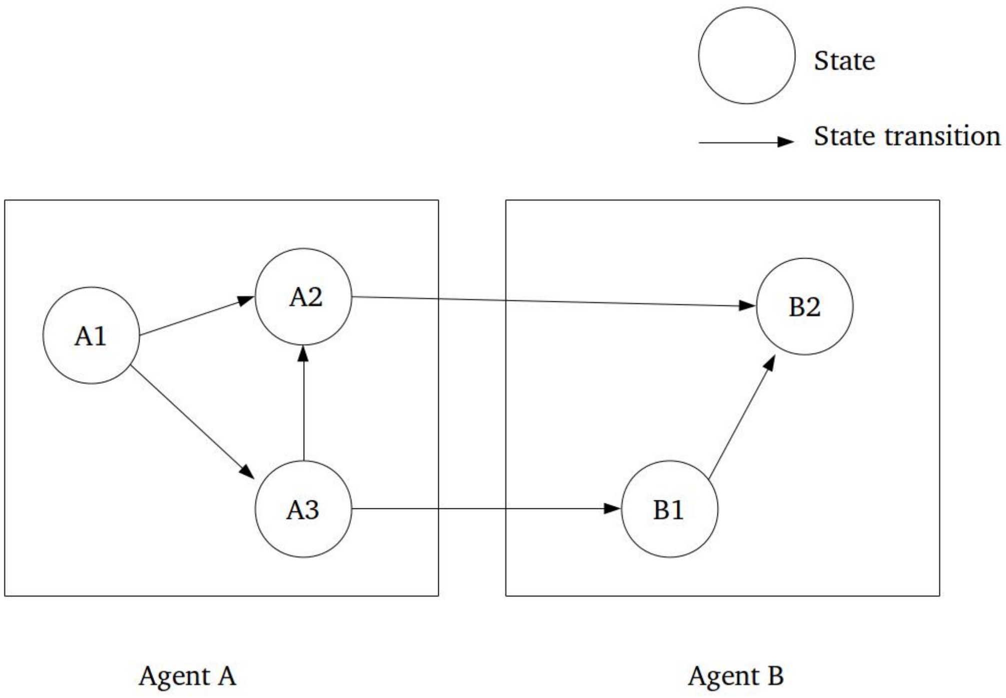


Figure 2.2: States and state transitions in a collaborative reinforcement learning environment.

In order to be able to decide which action to take, each state is associated with an estimation of the total reward, that will be received, when being in a state and following the rule to always choose the successor state that has the highest expected reward. This means, that each node has to store such a value for each of its internal and external states. The value of a state can be learned over time. Each neighbour calculates this value based on the values of the succeeding states, the reward that is received by performing the state transition to the succeeding states and the old value of the state.

The calculated value of a state will be sent to neighbour agents, so that these agents learn about values of external states. In the paper, this is called *advertisement* because each node advertises its local state values.

Although we can not use the reinforcement learning approach in the paper directly, because the number of states for each node is infinite (when each state is defined by “holding a certain query in the space” and there are an infinite number of such queries), we have used the idea of collecting feedback values for exploration queries that are estimations of the confidence with that a query can be answered via a neighbour. A node can use its collected feedback values to determine an estimation of confidence with that the query can be answered via itself. This estimation can be the basis for further feedbacks for some neighbours. With this method, knowledge about the ability to answer queries can spread through the network.

3 System Model and Problem Formulation

In this chapter, we introduce assumptions about the network and its participants. After this, we define the local data models and the notion of confidence. The chapter ends with a specification of the routing problem.

3.1 Network

We assume a network of nodes $p_1, p_2, ..., p_n$ in a mobile environment that can communicate with each other either directly or indirectly via a routing path. A node can be any device that is able to run the protocol. Because of the mobile environment, the network can be very heterogeneous and dynamic. Nodes may join and leave the protocol and may crash and recover at any time. We do not assume any upper bounds for message delivery, clock drift and process execution. Messages can be duplicated and nodes may receive old messages.

Further, we assume an undirected, acyclic topology. We write $\{p_1, p_2\}$ to indicate that there is a link between p_1 and p_2. The decision for an undirected topology has practical reasons: we will see that neighbours exchange queries and routing information in our protocols, so they know each other in any case. This results naturally in a bidirectional information flow. We assume acyclicity, because cycles in the network disturb routing decisions and the exchange of routing information in our protocols. Maintaining an acyclic topology under dynamic condition is a well-studied problem, see for example [CCK88].

In the following, we bring into focus an individual participant of the protocol.

3.2 Data Model

Each node p_i maintains a **data model** DM_i, where it can store data vectors. Data vectors in a metric space can have different dimensions and different dimensionality. This is because each dimension can be seen as variable that measures the value of one specific property of individuals. For example, if the data model contains samples of persons, the dimension “body height” measures the size of a concrete person. But there can be other dimensions for that same individual, like “monthly income” or “weight”.

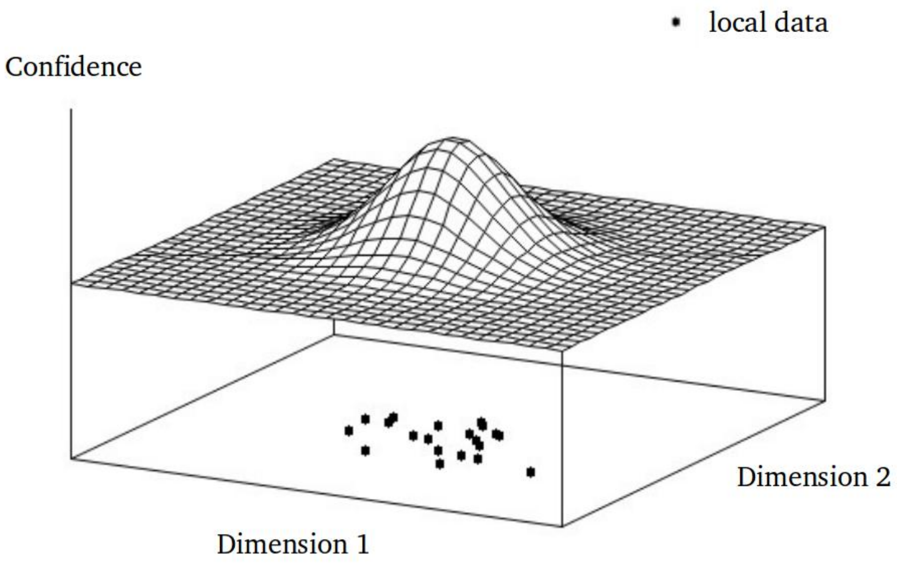


Figure 3.1: Confidence in local data.

We denote a data vector with $\mathbf{v}^{D'}$, where D' is the set of dimensions that are associated to the data vector. Assuming a set of possible dimensions $D = \{0, 1, 2, ..., m\}$, the set D' will be a subset of D. An example would be the data vector $\mathbf{v}^{\{0,1,2\}}$ that is a vector in the space $\mathbb{R}^3$ with dimensions $0, 1$ and 2. The set $\{0, 1, 2\}$ is a subset of D.

Together, the data models build the basic knowledge of the overall system. Dependent on the data model, nodes can have different **confidence** in their data. A node should have more confidence in its data in an area, where it has very much knowledge. We have to come up with a metric that measures the quality of data with respect to a certain query. This metric will be called *confidence score*. Assume a set of data vectors c that build a cluster in the space and a query $\mathbf{q}$. We have to come up with a measure $C(c, \mathbf{q})$ that measures the goodness of c for a query $\mathbf{q}$. We define the local confidence $C_i(\mathbf{q})$ of a node p_i for a query $\mathbf{q}$ to be the maximal confidence value for each data cluster in the data model:

$$C_i(\mathbf{q}) = max_{c \in DM_i} C(c, \mathbf{q})$$

In Figure 3.1 we visualize how the confidence metric could look like for a local data model, that consists of two-dimensional data vectors in the space. These vectors are the learned knowledge on a certain node. Each point in the space is associated with a confidence value, that gives information about how much knowledge a node has in this area. Areas in the space, where there are more information should have higher confidence. We can see, that the farther away from the local data vectors, the smaller the confidence score. The measure of confidence enables a node to extract its “area of expertise”.

From now on, we shift our focus away from the individual participant to the interaction between participants.

3.3 Query Routing

In Figure 3.2, we can see an example topology and the local data models of nodes p_0, p_1, p_2. Node p_0 receives a query, but finds that it has not enough knowledge for the query. So the query is routed through the network until node p_3 finds that it can answer the query locally with its local confidence.

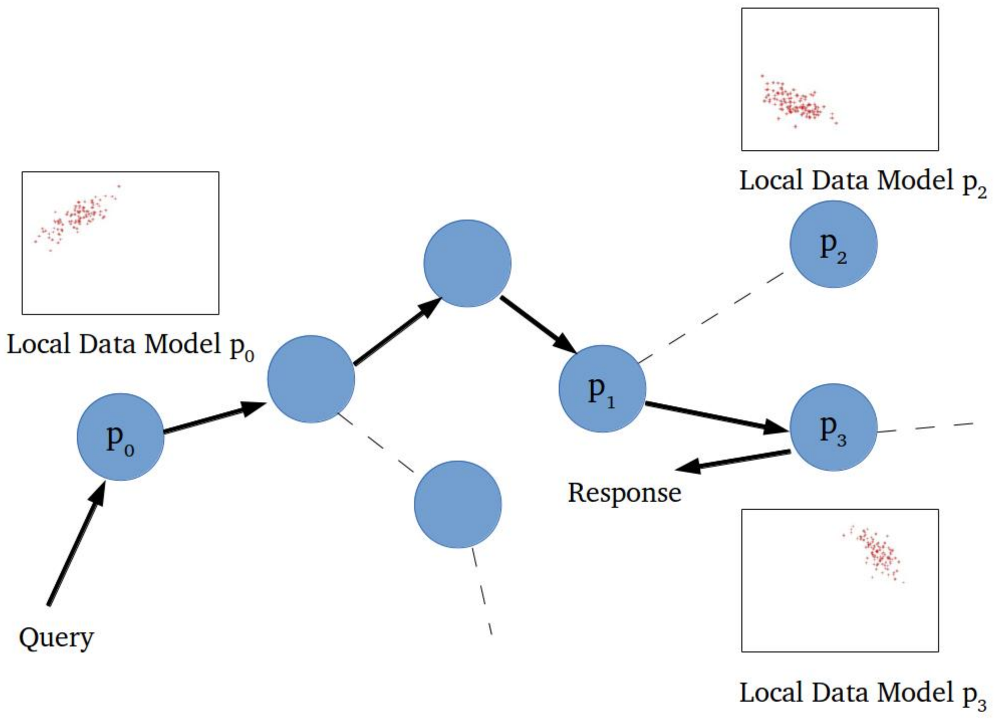


Figure 3.2: Query Routing

In the following, we give a problem statement for the routing.

Given: A query $q = (\mathbf{q}, C^{\mathbf{q}}_{min})$ on a node p_1. The query q consists of a query vector $\mathbf{q} \in \mathbb{R}^{m_\mathbf{q}}$ and a confidence threshold $C^{\mathbf{q}}_{min}$, with that a node should be able to respond the query. The threshold is more an indicator how hard the system tries to find a good confidence value. If no local confidence value is good enough, the system should retrieve the best confidence value. Note, that the dimensionality of the query could be different to the dimensionality of each local data model of a node on the way.

Find: The best neighbour to which the query should be forwarded. Assume $\hat{p}$ is the node with the highest local confidence for query $\mathbf{q}$. Then, the best neighbour is the one that lies on the path to $\hat{p}$. If each node is able to find the best neighbour, routing will be optimal, because the system will retrieve the best local confidence value for each query.

To maintain scalability, each node may only have a local view on the system. This means that it only knows its local data, its neighbours and some routing information to be able to route queries through the network, in order to find a node that has good fitting data in its local data model.

Our main goal in the thesis is to develop scalable routing protocols for multidimensional queries so that queries can be routed to nodes that maintain data with high confidence for the queries.

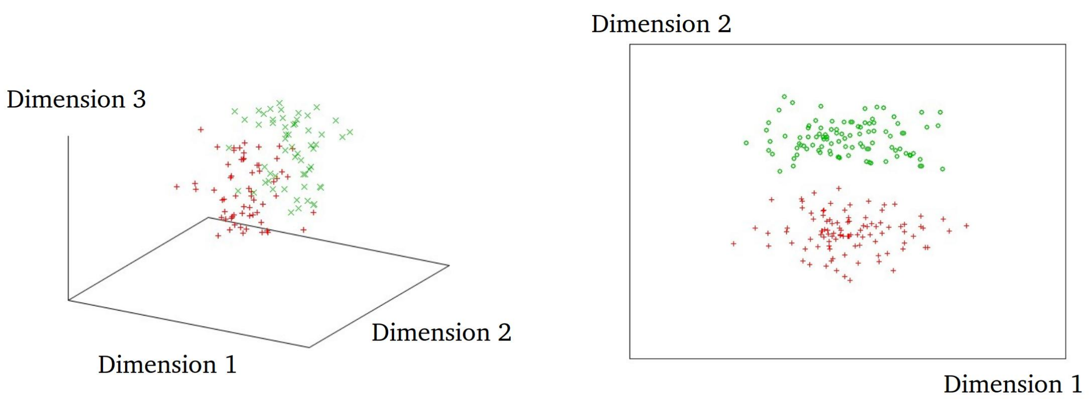
After introducing the confidence score, we can easily determine which data should have been retrieved by the system assuming an optimal routing protocol: the data with the highest confidence score for the query. But in large systems we can not implement an optimal routing protocol due to the necessity of global knowledge. Therefore, we have to come up with a scalable routing protocol that routes the query to nodes that can answer the query better.

When we assume very high dimensionality of the data, we have to solve another difficulty: the *curse of dimensionality* [VF05]. In higher dimensions data becomes sparse and the clustering property of data can become invisible. See for example Figure 3.3. We have plotted two clusters in the three-dimensional space and in the reduced two-dimensional space. The reduction makes the two clusters more visible. As we rely on clustered data for computing the confidence score, we have to develop methods to reduce the irrelevant dimensions and integrate dimensionality reduction into our routing approaches.

To summarize, the main challenges are the following:

- As the dimensionality of data in the data model can become very high, we have to come up with a method for dimensionality reduction in the data model to address the curse of dimensionality.
- Each node has to be able to determine the confidence in its local data for a given query. We have to define a measure for this purpose.
- We have to come up with a scalable routing protocol so that the system is able to route a query to a node having a high confidence value for the query.

- The routing protocol should work in the presence of heterogeneous dimensionality of the local data models in the system.



(a) Data clusters in three-dimensional space. **(b)** Data clusters in two-dimensional space.

Figure 3.3: Effects of clustering property, when reducing dimensionality.

4 Confidence-based Retrieval

In this chapter, we develop a very central concept of the thesis: the **confidence** in the data. Initially, we explain the **notion of confidence**, where we we describe "confidence" in words. In our approach, the determination of confidence is based on clusters. Therefore, we explain how the **clustering of the data** can be done. Afterwards, we introduce the **measure for confidence**, that is a concrete formula, instead of a concept. Given a cluster c and a query $\mathbf{q}$, the measure for confidence is based on the **cluster quality** of cluster c and a **distance metric** between cluster c and query $\mathbf{q}$. We develop formulas for these two concepts at the end of this chapter.

4.1 The Notion of Confidence

A common precondition for algorithms in the area of machine learning is to provide a data set, so that some knowledge can be learned. This data set has to be big enough, so that actually some valuable information can be extracted out of the data. If the learned knowledge is based on very much data, we should have more *confidence* in it, as if the underlying data set was small. We have to come up with a metric, that maps each point in the space to a confidence score that measures the quality of data for this point. Beside the *number of data,* confidence should be influenced by the density and the distance between data and query point.

We have already mentioned in the problem formulation (see Chapter 3), that we are interested in a system that responses queries with high-confident data. The provided data should reflect some knowledge in the area of the query. The system should return not only the *best* data, but also the confidence score of the data for the query. If the system has very little knowledge about a query, it will also provide this information to the user of the system by means of the confidence score.

Definition 4.1.1 (Confidence)
Confidence is a quality measure for a set of multidimensional data and a query.

In this chapter we define properties of this measure and develop a *confidence function* C, that fulfils these properties.

We assume the perspective in Figure 4.1.

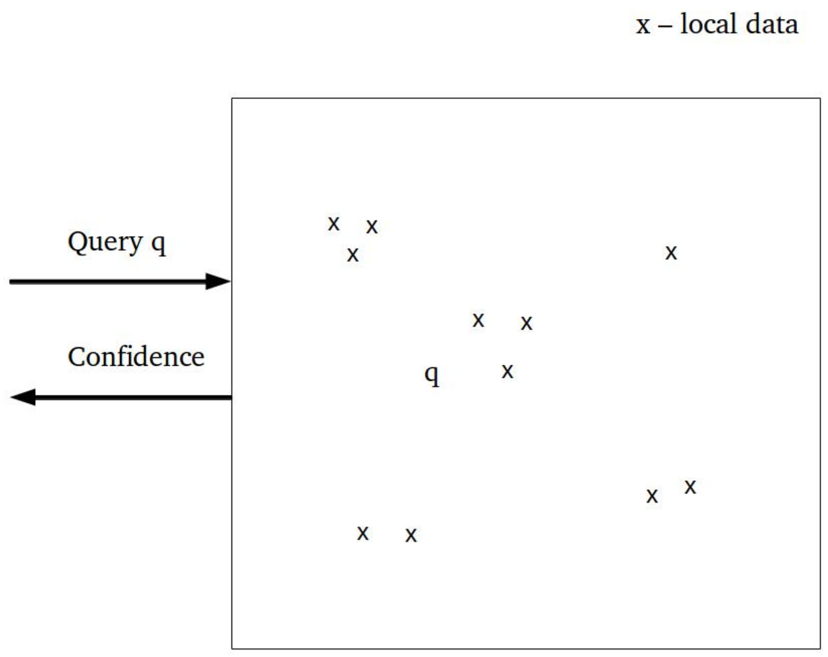


Figure 4.1: Retrieval of confidence score for a query on a node.

Each node maintains a local data model (see Chapter 3), that consists of a data set $V = \{\mathbf{v}_0, \mathbf{v}_1, ..., \mathbf{v}_t\}, \mathbf{v}_i \in \mathbb{R}^m$. When a node receives a query, it has to find out the confidence score of its local data V for a query $\mathbf{q} \in \mathbb{R}^m$. Hence, the data model maps an incoming query to a confidence score.

In summary, the measure of confidence gives us the opportunity to compare two data sets, to find out, which one has higher data quality for a certain query. In order to determine the confidence score for a data set, we divide the data set into clusters and compute the score separately for each cluster-query pair.

4.2 Clustering of the Data

The first step towards the computation of the confidence score is to cluster the data set. For clustering, we use the G-Means algorithm (see [HE03]) that performs multiple instances of K-Means clustering, but determines k (the number of clusters) automatically. After clustering the data, we can compute a confidence score for each cluster-query pair. The confidence score $C(c, \mathbf{q})$ for an arbitrary query $\mathbf{q}$ and a cluster c will consist of two distinct measures: the **quality of the cluster** itself and a **distance measure** between the query and the cluster centre. A cluster of high quality in proximity to the query will receive a higher score than a cluster of low quality that is far away. We can also imagine a situation, where the quality of one cluster is compensating the slightly greater distance (see Figure 4.2).

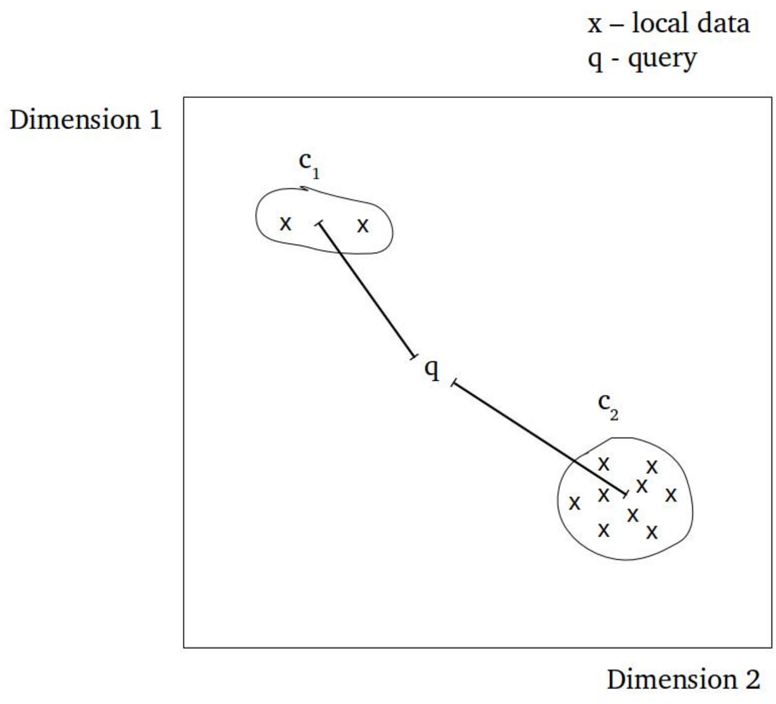


Figure 4.2: Cluster quality and query distance

The figure shows two data clusters c_1 and c_2. One is obviously of very good (c_2), the other of very bad quality (c_1). The query is somewhere in between the cluster centres, but a little closer to the bad cluster c_1. Nonetheless, we should take into account, that cluster c_1 consists of a very low number of points with a high variance. We should be more confident in learned knowledge based on cluster c_2 than in knowledge based on cluster c_1.

4.3 A Measure for Confidence

In this section, we develop a formula for computing the confidence score. For this purpose, we combine the fundamental requirements for a good score: The **quality of the cluster** should be good and the **distance between query and cluster centre** should be low. We investigate both requirements and define the confidence score to be the product of the quality of the cluster $Q(c)$ and the distance between query and cluster centre $Dist(c, \mathbf{q})$.

$$C(c, \mathbf{q}) = Q(c) * Dist(c, \mathbf{q})$$

The score will give a high value, if the **number of data vectors** in the cluster is high, the **variance** is low and the **distance** from the query to the centroid of the cluster is low.

4.4 Cluster Quality

We have to come up with a formula, that measures the quality of each cluster c. The quality score $Q(c)$ should increase with the number of data vectors and decrease, when the variance of the cluster increases (see Figure 4.2 in the last section).

We denote the number of data in the cluster as t_c, the variance as σ^2 and the centroid of the cluster as $\bar{c}$:

$$Q(c) = \frac{t_c}{\sigma^2} \qquad \sigma^2 = \frac{1}{t_c} \sum_{\mathbf{v} \in c} (\mathbf{v} - \bar{c})^2 \qquad \bar{c} = \frac{\sum_{\mathbf{v} \in c} \mathbf{v}}{t_c}$$

The quality of cluster c is the number of data vectors in c divided by the variance of the cluster. This satisfies our requirements for the quality score directly. A high number of data vectors in each cluster will result in a large quality score. On the other hand, increasing the variance of the cluster, decreases the quality score.

4.5 Distance Metric

To be able to compute the confidence score, we still need a measure for distance between the query and the centroid of vectors in the cluster. We have decided to include the evolution of the distance function in this thesis, because we think that it can give more insights to the reader than only presenting the final metric and its evaluation.

In the following, we introduce some properties, that our distance metric should fulfil.

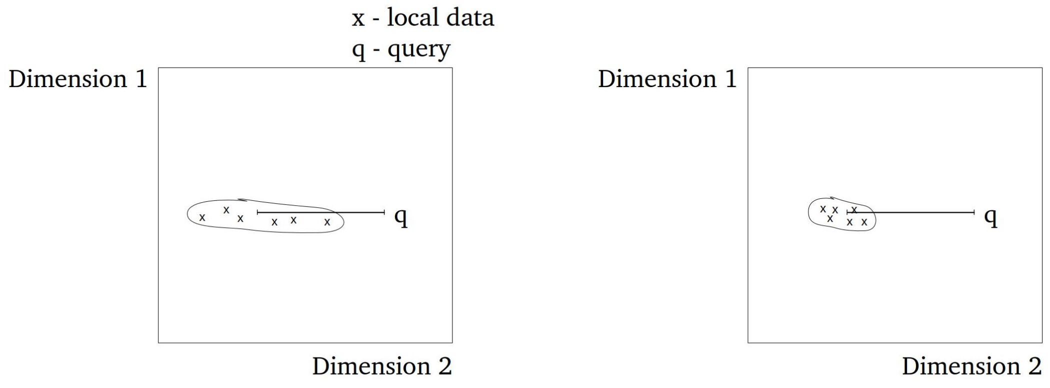
Property 4.5.1
The distance metric $Dist(c, \mathbf{q})$ should be high, if the euclidean distance $||\mathbf{q}, \bar{c}||$ from the query $\mathbf{q}$ to the centroid of the cluster c is low. On the other hand, if $||\mathbf{q}, \bar{c}||$ is high, $Dist(c, \mathbf{q})$ should be low.

This is the fundamental requirement on our distance function. Confidence of a query to a cluster should increase, when the euclidean distance to the cluster decreases. We want to reward short distances to the cluster.

The following property is not as apparent as property 4.5.1.

Property 4.5.2
When a cluster c has low variance, the decline of the score should be relatively high with increasing euclidean distance. On the other hand, if the variance is high, the decline of the score should be low.

To understand the meaning of this property, see Figure 4.3.



(a) High variance should result in a high distance score

(b) Low variance should result in a low distance score

Figure 4.3: The variance of a cluster influences the distance function

We can see two equal sized cluster with different variances and a query that has the same distance to the centroids of the clusters. Intuitively, the distance score should be higher for the cluster with higher variance. This cluster fits much better to the query. Hence, the decline of the distance score should not only depend on the euclidean distance, but also on the variance of the cluster.

Before introducing the final distance metric, we will investigate the inverse euclidean distance function.

4.5.1 Inverse Euclidean Distance

Initially, we have chosen a similar formula that fulfils property 4.5.1. The **inverse of euclidean distance** is defined as:

$$Dist''(c, \mathbf{q}) = \frac{1}{||\bar{c} - \mathbf{q}||}$$

The distance score increases, when the distance between the query $\mathbf{q}$ and the centroid of the cluster $\bar{c}$ decreases. This satisfies property 4.5.1. We have plotted the function in Figure 4.4.

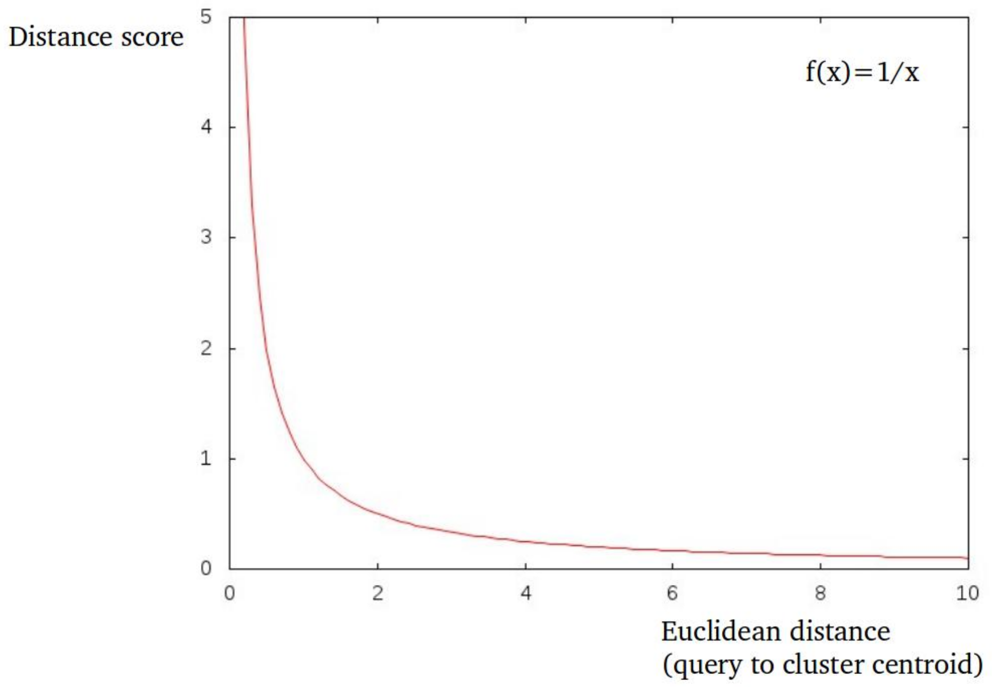


Figure 4.4: Inverse euclidean distance function.

The horizontal axis represents the euclidean distance between a query and the centroid of a cluster. The vertical axis gives the associated distance score.

As we can see, the inverse euclidean distance function goes against infinity for positive very small distance values. That means, the difference between the scores of two queries, that are very near to the cluster centre can be unimaginably high. On the other hand, queries that are far away from the cluster centre still can have a reasonable score (fat tail). To be more precise, queries that have an euclidean distance to the cluster centre that is greater than two or three, have more or less the same distance value.

Assume a number of clusters $c_0, ..., c_t$ and a query $\mathbf{q}$. We want to determine the best cluster for the query, i.e., the cluster with highest $C(c, \mathbf{q})$ value. When there is no cluster, that is very close to the query, the confidence score will be more or less *independent* of the distance from the query to the cluster centre. This is because of the property of the inverse euclidean distance function, that it produces more or less the same value for all clusters.

Instead, the quality scores of the clusters dominates the distances. This results in a situation, where good clusters attract nearly all queries in the space. See Figure 4.5 for an illustration of this issue.

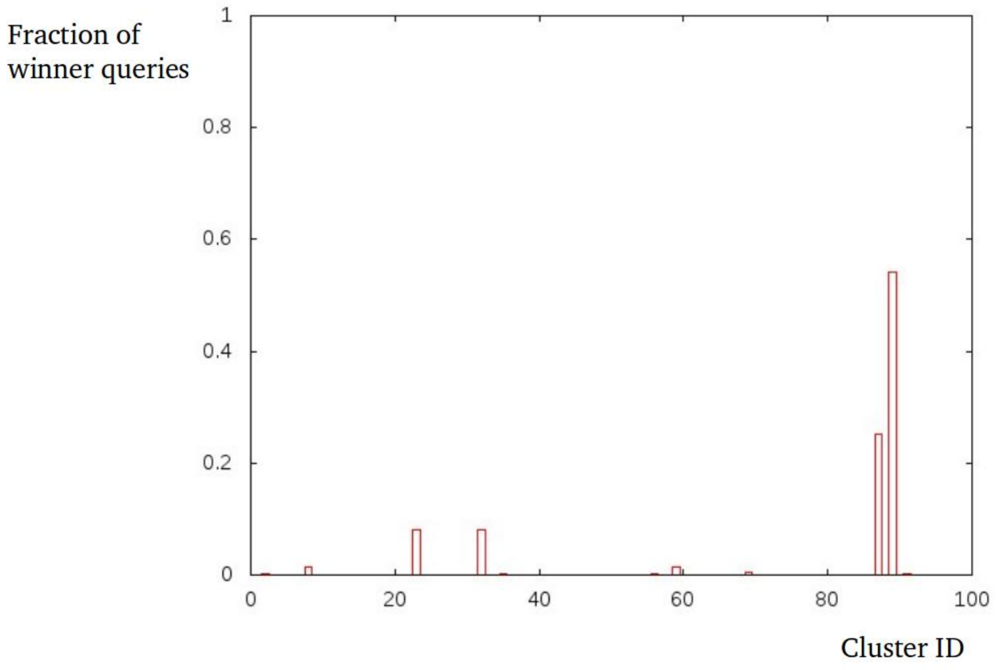


Figure 4.5: Best clusters for random queries with inverse euclidean distance function.

We have initialized 100 clusters with random means, random standard deviations $\sigma \in [1 : 100]$ and a random number of points in the interval $[2 : 100]$. Hence, the quality scores for the clusters differ widely. Then we have generated 10000 random queries and determined the best cluster for each query. For this, the confidence metric based on inverse euclidean distance was used.

For each cluster c we have plotted the fraction of queries, for which it was the best cluster, that means $c = argmax_{c_i} C(c_i, \mathbf{q})$.

There were only four clusters, that answer nearly all queries. This is because these four clusters have a very low variance and a relative high number of points. Here, the cluster quality dominates the distance score.

This issue is problematic as we have seen, but even more important, property 4.5.2 is not fulfilled either. The variance of a certain cluster has absolutely no influence on the distance score. We will solve these issues with a distance score that is based on the gaussian distribution function.

4.5.2 Gaussian Distance

We have seen, that the inverse euclidean distance does not fulfil property 4.5.2. We need a distance function that decreases with increasing euclidean distance between the query and the cluster. But the decreasing should be dependent on the variance of the cluster. For this purpose, we will introduce a new distance metric that is based on the normal distribution.

The standard normal distribution with mean μ and standard deviation σ is defined as:

$$gauss(x, \mu, \sigma) = \frac{1}{\sigma * \sqrt{2\pi}} \exp^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Then, the distance score for a query $\mathbf{q}$ to the cluster centre $\bar{c}$ from cluster c is:

$$Dist'(c, \mathbf{q}) = gauss(||\bar{c} - \mathbf{q}||, 0, \sigma_c)$$

As standard deviation for the gaussian function, we take the standard deviation of the cluster σ_c. We will explain the reason for this in the following.

The function is plotted in Figure 4.6 for different standard deviations. In the figure, we compared two clusters c_1 and c_2 with standard deviations one and two. We can see, that property 4.5.2 is fulfilled automatically, when taking the standard deviation of the cluster as standard deviation of the gaussian function: The gaussian distance function will decline slowly, when the variance of the cluster is high. On the other hand, if the variance of the cluster is low, the gaussian distance function will decline much faster.

Also, if there is a dense cluster with small variance, a very close query will receive a relative high score. This is also a very intuitive property of data confidence. When we have a cluster of dense data in proximity of the query, we should be more confident in the query response than if the data in the cluster is sparse.

In Figure 4.7 we have plotted the gaussian distance function for a cluster. Each point on the upper plane corresponds to the distance score, a query would receive.

Queries, that are very close to the centroid receive the highest values. The score smoothes out for more distant queries and becomes eventually (practically) zero.

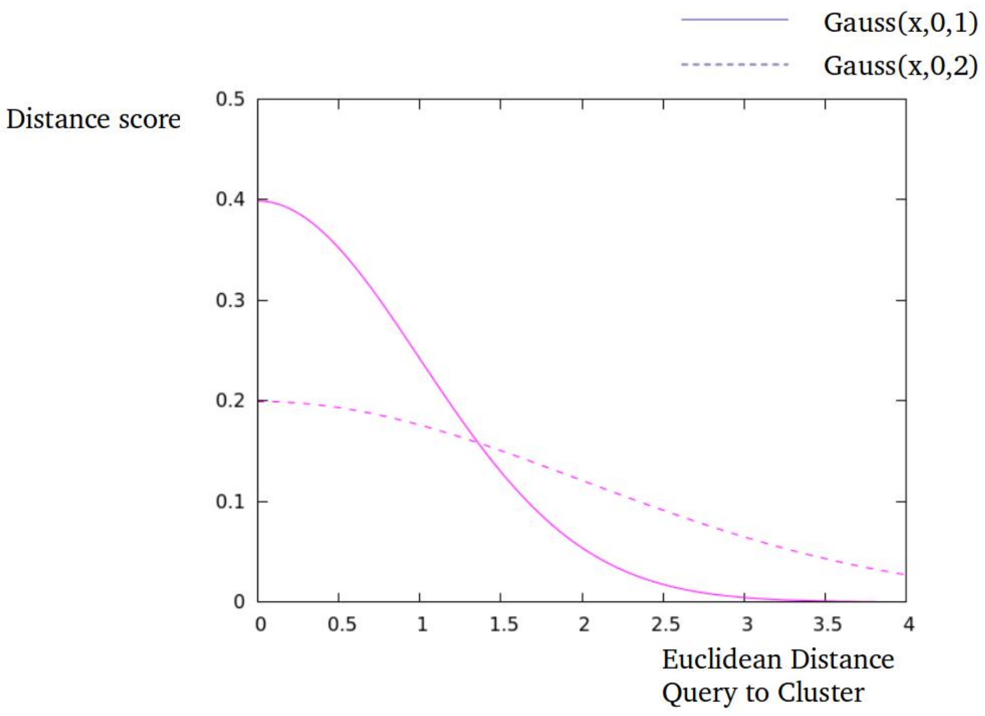


Figure 4.6: Gaussian distance function for standard deviation one (continuous line) and two (dashed line).

As we have seen, property 4.5.1 and property 4.5.2 are fulfilled, when using the gaussian distance function. Additionally, there is a bounded upper distance score for close queries and the exponential decrease of the function value makes the score for distant queries practically zero. Hence, only relative close queries get a reasonable distance score. This solves the undesirable balancing property with the inverse euclidean distance function. The quality score can not compensate for distant queries, as the distance score will be practically zero.

In Figure 4.8 we can see the positive implication of this:

We have repeated the experiment from the inverse euclidean distance function for the distance score that is based on the gaussian function. The balancing of the best clusters for random queries is much better. High quality clusters still attract more queries than low quality clusters. But the distance between the query and the cluster centre has also influence on the confidence score, so that the distribution of queries is not as extreme as above.

The gaussian distance score fulfils our requirements on a distance function. Nevertheless, we will extend it in the following to a *multivariate* gaussian distance function, so that we can work with multivariate distributed data clusters.

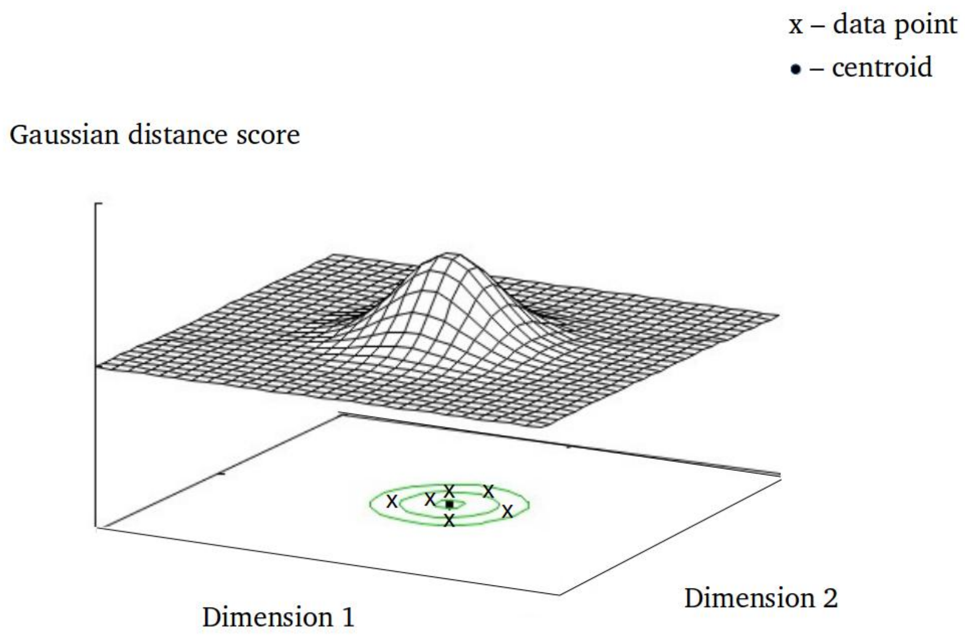


Figure 4.7: Gaussian distance function for all queries and a two dimensional data cluster.

4.5.3 Multivariate Gaussian Distance

Multivariate normal distributed clusters are a more general form of normal distributed clusters. The difference is, that each two dimensions can have a certain correlation. This results in clusters, that can be "squeezed" to some extend (see Figure 4.9).

The correlation between two dimensions can be specified in the **covariance matrix** Σ. The covariance matrix can be extracted out of the data. This can be done in three steps:

1. Data matrix $\mathbf{V}$: Assume, we have t data vectors $\mathbf{v}_1, ..., \mathbf{v}_t$, each having dimensionality m ($\mathbf{v}_i \in \mathbb{R}^m$. Then, the data matrix $\mathbf{V}$ is the matrix with the t vectors $\mathbf{v}_i$ as rows.
2. Deviation matrix $\mathbf{V}'$: We subtract from each element i, j of the data matrix $\mathbf{V}$ the mean value μ_j of dimension j. The mean μ_j can be extracted easily from the data set.
3. Covariance matrix Σ: $\Sigma = \mathbf{V}'^T \mathbf{V}'(1/t)$

The gaussian distance measure, that was introduced in the last section, is a special case of a more general kind of function: the **multivariate normal distribution**. In the following, we will introduce a distance measure, that can also be applied to multivariate data clusters.

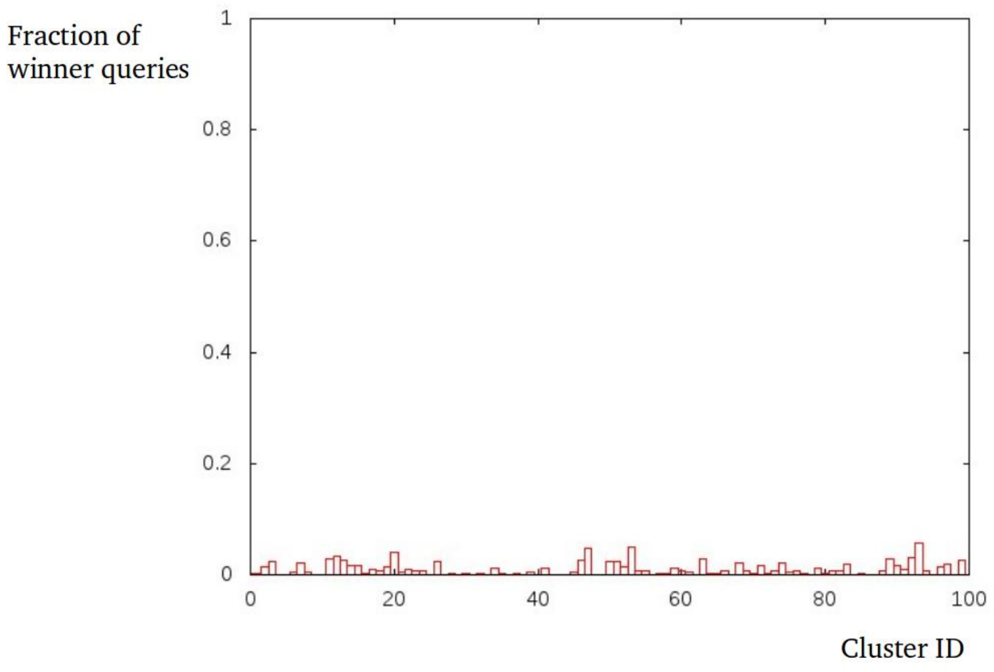


Figure 4.8: Best clusters for random queries with gaussian distance function.

The formula to compute the multivariate gaussian distance function for query $\mathbf{q} \in \mathbb{R}^m$ and cluster c is:

$$Dist(c, \mathbf{q}) = \frac{1}{(2\pi)^{0.5m}|\Sigma|^{0.5}} \exp^{-0.5(\mathbf{q}-\mu)^T\Sigma^{-1}(\mathbf{q}-\mu)},$$

where $|\Sigma|$ is the determinant and Σ^{-1} is the inverse of the covariance matrix Σ. The mean vector $\mu = (\mu_1, ..., \mu_m)^T$ can be extracted out of the data set.

The confidence measure for multivariate clusters is a more general form of the confidence score, that we have developed in the last section. If we would like to know the confidence value for a cluster with no covariances between any two dimensions, we will get the same value in the multivariate function.

In Figure 4.9 we give an example, where the two dimensions are more correlated. Queries that lie on the same circle, have the same distance score. This means, that a query $\mathbf{q}$ that lies more into the direction of the squeezed cluster than another query $\mathbf{q}'$ receives a higher score than query $\mathbf{q}'$, even if both have the same absolute euclidean distance to the cluster centre.

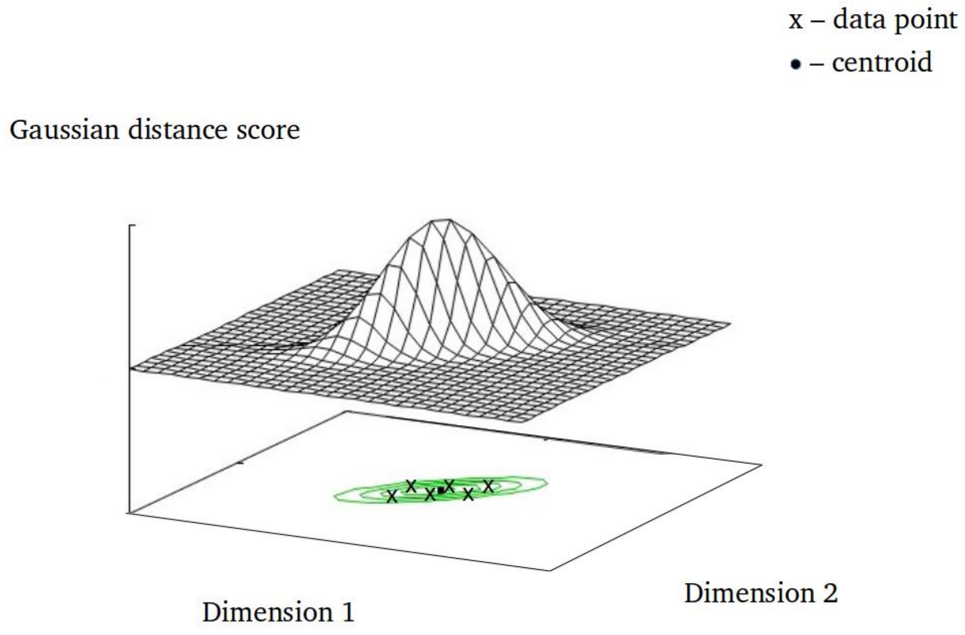


Figure 4.9: Multivariate gaussian distance function for all queries.

In summary, we have developed a measure for **confidence** in this chapter. The metric consists of two parts: the **cluster quality** and the **distance function**. The main part of this chapter was finding a good distance function. Because of its good properties, the **multivariate gaussian distance function** will be used to compute confidence values between queries and clusters. The question of how good a query fits to a cluster can be answered computationally by this measure.

With this, we have a practical instrument to determine, which data will be a good response for a query.

5 Routing Based on Data Aggregation

In this chapter, we present an approach for query routing, that is based on the aggregation and forwarding of data clusters. Data clusters can consist of a huge set of data vectors. In order to save bandwidth, when forwarding data clusters, we develop an approach how to make data clusters more compact. Nodes that receive such compact cluster information can regenerate data that is very similar to the original data cluster. The regenerated data vectors can be used as routing information, as we will see later.

We begin this chapter with a **protocol overview**. In Section 5.2, we look into the theoretical foundations of how to extract compact cluster information on a node, so that the information can be forwarded to a neighbour without sending each single data vector. A node receiving this compact cluster information has to regenerate the data. We present a method how this can be achieved in Section 5.3. Afterwards, we give a **protocol description** in pseudocode. Finally, we **discuss** some properties of the approach.

5.1 Protocol Overview

In Chapter 3 we have already mentioned, that each node maintains a **local data model** (DM). Additionally, each node holds for each direct neighbour a **local routing model** (RM), where information can be stored about the neighbours ability to answer the query.

Receiving a query, a node has to decide, whether to forward the query $q = (\mathbf{q}, C^{\mathbf{q}}_{min})$ (see Chapter 3) to a neighbour or to answer it locally. For this purpose, we have introduced the measure of **confidence** in Chapter 4: Each node is able to compute the score $C(c, \mathbf{q})$ for each data cluster c in the local data model. If the score is good enough, the query can be answered locally. This is the case, if there exists a local data cluster c, such that $C(c, \mathbf{q}) > C^{\mathbf{q}}_{min}$.

Otherwise, if the local data is not good enough for the query, a node will have to decide, to which neighbour the query should be forwarded. Therefore, we need to fill the local routing model for a neighbour with information about the data that is reachable via this neighbour. In Figure 5.1 we can see the process of forwarding clustering information into the routing models of node p_1.

We assume a very simple example, where node p_1 is connected to nodes p_2 and p_4. Additionally, node p_2 has a connection to node p_3. We have given the data models of nodes p_2 and p_4. Node

p_2 has already clustering information in its local routing model for node p_3. Initially the routing model of node p_1 is empty. Nodes p_2 and p_4 decide to send routing information to node p_1 about data that is reachable via themselves. Therefore, the nodes have to extract compact cluster information for all local clusters in their routing and data models. For each cluster they determine one such compact cluster information that can be send to node p_1. When node p_1 receives compact cluster information, it regenerates the data in the routing model of the sender. With this method, node p_1 learns about data that is reachable via its neighbours.

There is another detail, we want to mention here. Node p_2 has actually another routing model that is omitted in the figure: the routing model for node p_1. When node p_2 intends to send cluster information, it has to ignore clusters in this routing model, because node p_1 is already the source of all the cluster information in this routing model.

As we will see, the compact cluster information is actually compact and can be sent easily to neighbours. The compact cluster information is also sufficient to enable node p_1 to reproduce a cluster of local data, that is very similar to the original cluster on node p_2.

The reproduced data vectors can be stored directly in node p_1's routing model for node p_2. With these data vectors in the routing model, node p_1 can determine very easily an estimation of the confidence score $C_{est}(\mathbf{q}, c)$ for a received query $\mathbf{q}$ and a cluster c. Node p_1 will then forward the query to the neighbour, whose routing model contains the best cluster, if it can not answer it locally.

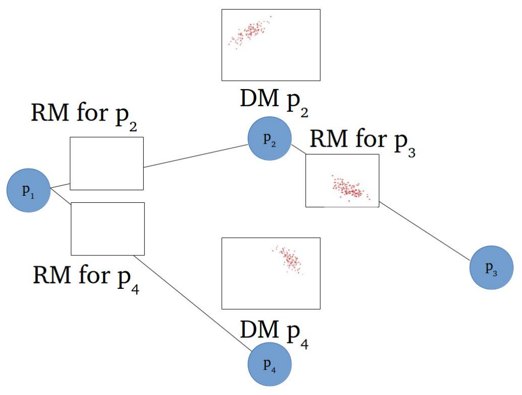
In the following, we show, how a node can determine the compact cluster information, it intends to forward to a neighbour. Afterwards, we introduce a method to generate data out of this cluster information.

5.2 Analysis of Multivariate Clusters

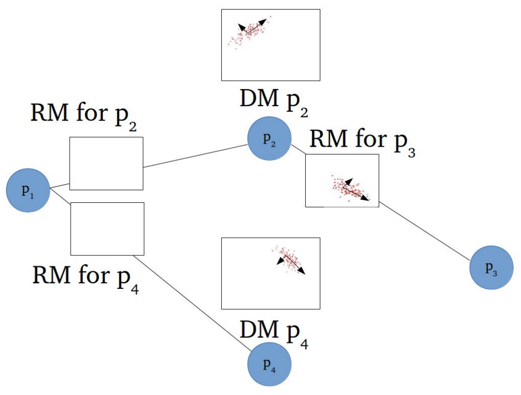
We assume, node p_2 wants to forward cluster information to node p_1. For this purpose, node p_2 clusters the data in its local data model with the G-means algorithm (see Chapter 4). Node p_2 has to forward data that is reachable via itself. Because of that, it clusters the local routing models as well and merges both sets of clusters into the set $S = \{c_1, c_2, ...\}$.

This is basically the information, node p_1 is interested in. But it would be problematic to simply send the set S to node p_1, because the individual clusters can consist of a huge number of data vectors. A single routing cluster on node p_2 can actually be a merged super-cluster of all reachable data via a neighbour of node p_2. Therefore, it makes sense to model each individual cluster $c_i, i \in \{1, 2, ..., |S|\}$ as multivariate data cluster c_i':

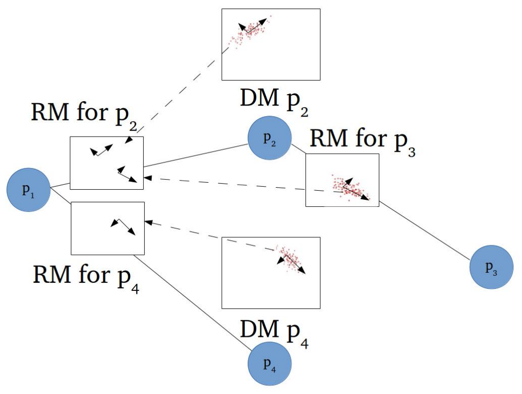
$$c_i' = (\Sigma_{c_i}, \mu_{c_i}, t_{c_i}),$$



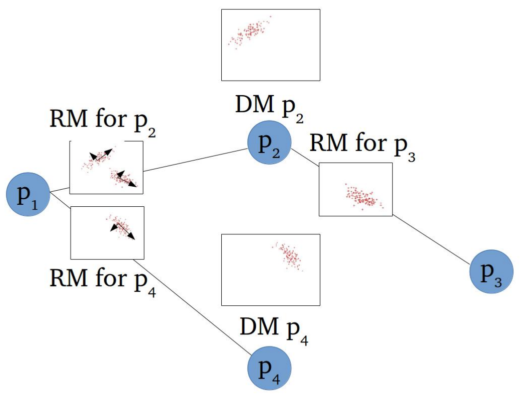
(a) Initially, the routing models of node p_1 are empty. Node p_2 already has some information in the routing model for node p_3.



(b) Nodes p_2 and p_3 intend to forward cluster information to node p_1 and analyze their local clusters.



(c) Nodes p_2 and p_3 send the analyzed, compact cluster information to node p_1.



(d) Node p_1 regenerates the data vectors out of the cluster information and puts the regenerated data into its local routing models.

Figure 5.1: Example of forwarding cluster information

where Σ_{c_i} is the covariance matrix of cluster c_i, μ_{c_i} is the mean vector or centroid and t_{c_i} is the number of data in cluster c_i. See Chapter 4 for how to determine the covariance matrix and the mean vector of a data set.

The list of compact cluster information that p_2 can send to p_1 is now:

$$S' = \{c'_1, c'_2, \ldots\}$$

In contrast to the set S, the set S' is very compact, because each individual component is independent of the number of data vectors it encodes.

We will look now into a method for node p_1 to regenerate data out of the list S'.

5.3 Generation of Multivariate Clusters

We assume, that node p_1 has received a list of cluster information S' from node p_2. Node p_1 can then use this information for its routing model for node p_2. This can be done by regenerating the cluster c' for each cluster $c' \in S'$. In [Her98], a method is given to draw a random vector according to a multivariate normal distribution that is given by a $(m \times m)$ covariance matrix Σ and the mean vector $\mu \in \mathbb{R}^m$:

1. Generate a random vector $\mathbf{v} = (\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_m)^T$, so that each $\mathbf{v}_i$ is independently generated by a normal distribution with mean $\mu = 0$ and variance $\sigma^2 = 1$. It is important that the individual $\mathbf{v}_i$ are independent. If so, this is already a possibility to draw values from the multivariate distribution, where the mean is $\mathbf{0}$ and the covariance matrix is the identity matrix (the diagonal values equal one and the rest equals zero).
2. Afterwards, the identity matrix has to be transformed into the covariance matrix Σ. This can be done by finding the $(m \times m)$ matrix Φ that has the normalized eigenvectors of Σ as columns, as well as the matrix Λ that has the eigenvalues as diagonal entries that belong to the eigenvectors in Φ. There are many methods to calculate the eigenvectors and eigenvalues of a matrix like for example the *eigendecomposition* of a matrix.
3. Let $\Omega = \Lambda^{0.5}\Phi$. Then we can transform the vector $\mathbf{v}$, such that it would be produced by a multivariate normal distribution with mean μ and covariance matrix Σ:

   $$\mathbf{v}' = \Omega\mathbf{v} + \mu$$

This can be seen, as deforming the normal distributed cloud of produced vectors to a multivariate distribution with mean $\mathbf{0}$ and translating this cloud to the new mean μ. With this, we are now able to generate random vectors out of a multivariate normal distribution, that is given by the compact cluster information.

5.4 Protocol Description

In this section, we will give algorithms, that describe how the routing and information forwarding protocol works. Each node maintains the following variables:

- A set of adjacent nodes, called $neighbours$.

- The $dataModel$ can be seen as module that manages the local data and offers a possibility to calculate the confidence score for each cluster. We define the best local cluster to be $\hat{c}$. The response vector $\hat{\mathbf{v}}$ will be the best local clusters centroid:

  $$\hat{c} = argmax_{c \in dataModel} C(c, \mathbf{q}) \qquad \hat{\mathbf{v}} = \frac{\sum_{\mathbf{v} \in \hat{c}} \mathbf{v}}{t_{\hat{c}}}$$

- A set of routing models $routingModels$, one routing model for each neighbour. In this approach, a routing model is much like a data model, we will save a number of local data vectors in it and determine the estimated confidence for queries to find out, which neighbour is likely to answer the query best.

Algorithm 5.1 Node p_i receives query message

```
1:  set $neighbours$
2:  module $dataModel$
3:  set $routingModels$
4:  on receive(QUERY, $q = (\mathbf{q}, C^{\mathbf{q}}_{min})$, $visited$) from node $p_j$ do :
5:      if $C^{\mathbf{q}}_{min} < C_i(\mathbf{q}) \vee neighbours \setminus \{p_j\} = \emptyset$ then
6:          send(RESPONSE, $\hat{\mathbf{v}}$, $C_i(\mathbf{q})$, $q$, $visited$) to node $p_j$
7:      else
8:          $visited \leftarrow visited + [p_i]$
9:          send(QUERY, $q = (\mathbf{q}, C^{\mathbf{q}}_{min})$, $visited$) to $bestNeighbour(\mathbf{q})$
10:     end if
11: endOn
```

In Algorithm 5.1 we show the behaviour of a node p_i on receiving a query from node p_j. Node p_i will answer the query q, if either its local data model possesses data that has high enough confidence or if it has no other node to forward the query to. Otherwise, it will forward the query to the neighbour, whose routing model gives the highest estimated confidence for the query.

Algorithm 5.2 Node p_i receives response message for a query

```
1: on receive(RESPONSE, $\mathbf{v}$, $C$, $q$, $visited$) from node $p_j$ do :
2:     if $C_i(\mathbf{q}) <= C$ then
3:         send(RESPONSE, $\mathbf{v}$, $C$, $q$, $visited - [p_i]$) to $visited.previous()$
4:     else
5:         send(RESPONSE, $\hat{\mathbf{v}}$, $C_i(\mathbf{q})$, $q$, $visited - [p_i]$) to $visited.previous()$
6:     end if
7: endOn
```

When a node decides to answer the query locally, the response associated with the confidence value will be send backwards along the path, the query has taken. If a node receives a response (see Algorithm 5.2) and can answer the query locally with a higher confidence value, it will replace the received response with its own and adapt the confidence value for the next receiver.

Regularly, each node should initiate a cluster forwarding round (Algorithm 5.3). Assume node p_i wants to send updates to node p_j. First, it analyzes its local data clusters and all its local routing clusters, except those in the routing model for node p_j, and determines the set S' like described in the previous sections. It then sends S' to node p_j, which regenerates the data in its local routing model for node p_i.

Algorithm 5.3 Node p_i: cluster forwarding

```
1: on data forwarding event for node $p_j$ do :
2:     if received updates from each $p \in neighbours \setminus \{p_j\}$ then
3:         send(UPDATE, $S'$) to node $p_j$
4:     end if
5: endOn
6: on receive(UPDATE, $S$) from node $p_j$ do :
7:     Generate cluster in routing model from $p_j$
8: endOn
```

6 Routing Based on Query Learning

In large and dynamic networks, the DA approach has some problematic properties. Data models can evolve continuously and nodes can join and leave the system very frequently. In this setting, maintaining routing models, that are up to date, is very expensive, because each change in a data model triggers the whole data aggregation process in which many nodes are involved.

Therefore, we present the **query learning** (QL) routing approach that learns the routing of queries with the help of past queries. In contrast to the DA approach, the learning is done in smaller portions, i.e., queries, instead of clusters of reachable data.

Initially, we give a detailed **protocol overview**. This is followed by a **protocol description** using algorithms in pseudocode. The chapter ends with an approach to **reduce dimensionality** of the local models and a **discussion** of the protocols properties.

6.1 Protocol Overview

We start this section with a high-level description of how routing can be performed based on the routing models on a node. Then, we investigate in detail, how exactly a single routing model works. Afterwards, we introduce the different possibilities to maintain routing information in the routing models on a node.

6.1.1 Routing of Queries

When a participant p_1 of the system receives a query $q = (\mathbf{q}, C^{\mathbf{q}}_{min})$, where $\mathbf{q} \in \mathbb{R}^m$ and $C^{\mathbf{q}}_{min}$ is the minimal expected confidence value, node p_1 has to decide, whether to answer the query locally or to forward the query to some neighbour (see Figure 6.1).

Node p_1 answers the query locally, when the local confidence $C_i(\mathbf{q})$ exceeds the minimal confidence score $C^{\mathbf{q}}_{min}$ or there is no other neighbour to forward the query to. Then, a node will simply determine the local confidence score for the query based on the data in its local data model (see Chapter 4) and send a response back to the last query issuer.

If it can not answer the query locally, node p_1 will have to forward the query to a neighbour. In routing, there often exists a routing table, that maps a query to an outgoing link. Because of

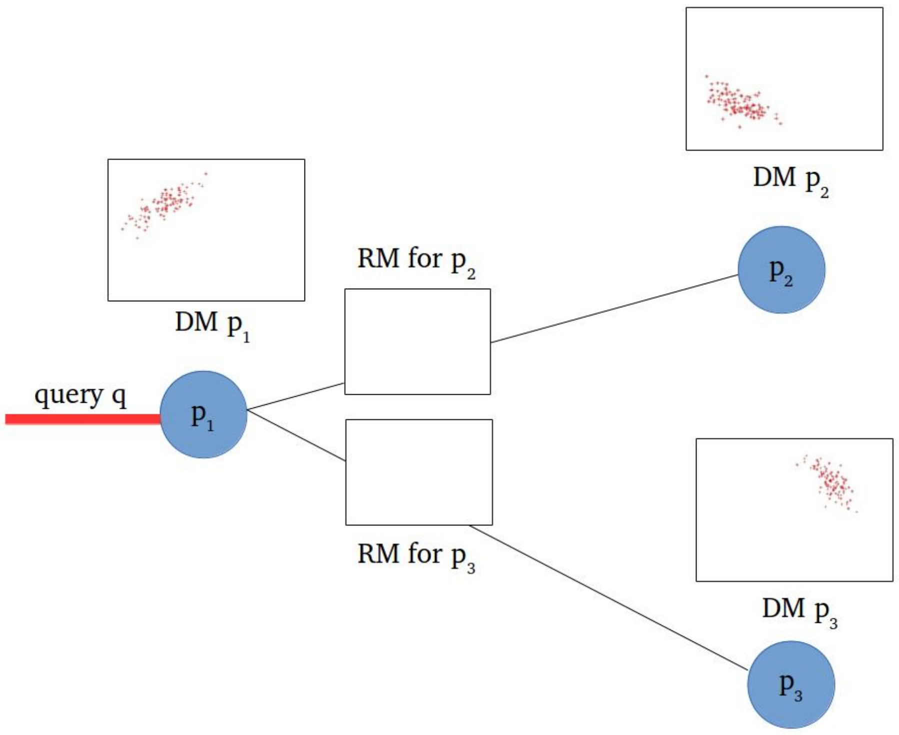


Figure 6.1: Routing decision on a node p_1. The data models (DM) contains local data, each neighbour is represented locally with a routing model (RM). Where to route the query?

the infinite number of possible queries, we can not rely on an exact routing table. Instead, we try to come up with a routing model for each outgoing link. This routing model models the "area of expertise" following the associated link. Technically, a node can ask each of its routing models for an estimation about the confidence value that would be retrieved, when the query was handed to the neighbour that is represented by the routing model. We call this function CONFIDENCEESTIMATION($\mathbf{q}$). The query is then forwarded to the neighbour for that the highest confidence estimation was retrieved.

6.1.2 Query based routing model

Subsequently, we show how the confidence estimation CONFIDENCEESTIMATION($\mathbf{q}$) can be computed in the routing model.

Node p_1 maintains a routing model for neighbour p_2. The routing model consists of query-confidence pairs, that it has learned during the protocol execution. We will see later how this is done exactly. We denote a query-confidence pair as $(\mathbf{q}_i, C_i)$. Each such pair represents an experience or an estimation of the confidence value that will be retrieved, when forwarding the query to node p_2.

See Figure 6.2 for an example of a routing model. The routing model consists of two dimensional data. Each query-confidence pair represents a piece of knowledge, namely the confidence value that can be expected for a query on this point in the space. For queries that are different to those in the model, we try to predict confidence values using the known queries. This prediction is actually the confidence estimation CONFIDENCEESTIMATION($\mathbf{q}$).

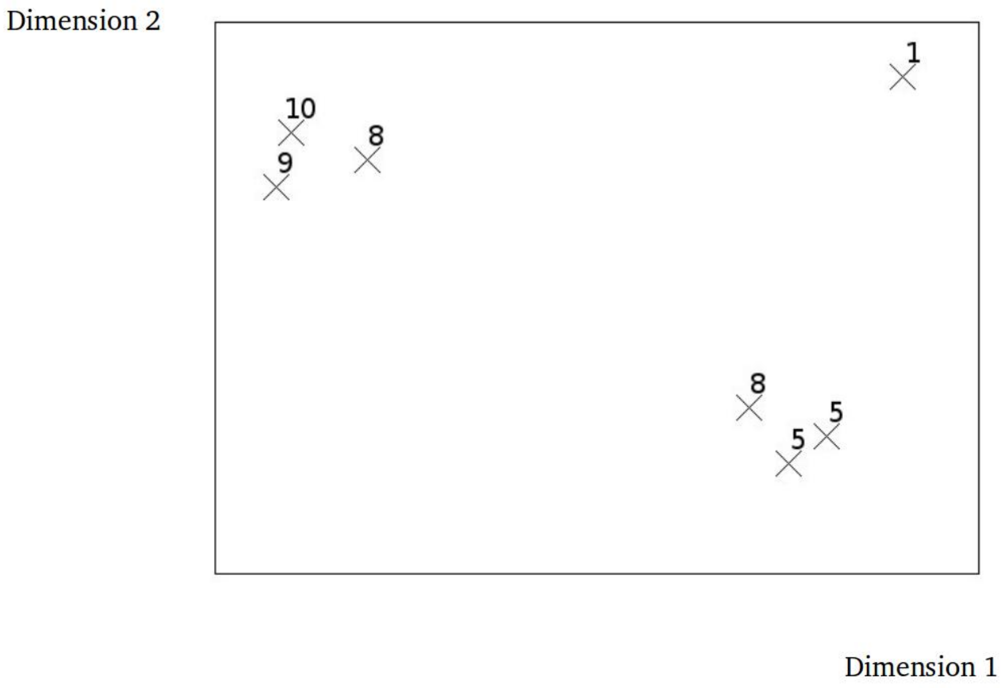


Figure 6.2: Routing model filled with query-confidence pairs.

The estimation of confidence will be based on clusters of similar query-confidence pairs. For clustering we use the weighted k-means clustering algorithm (compare [CYTZ09]). The difference between k-means and weighted k-means clustering is, that in weighted k-means vectors can have different influence on the clustering (higher weights means higher influence). As weights, we use the confidence values of the queries. This has the effect of practically ignoring very bad queries in the space.

In Figure 6.3 we can see the same routing model as above with clustered queries. There are two clusters of query vectors, one is plotted with empty circles, the other with empty squares.

For the clustering we have used weighted k-means clustering. Each cluster c has a weighted centroid $\bar{c}_{weighted}$ that is given by a filled circle and a filled square. As we can see, the bad query with confidence 1 has little effect on weighted clustering and the weighted centroid computation.

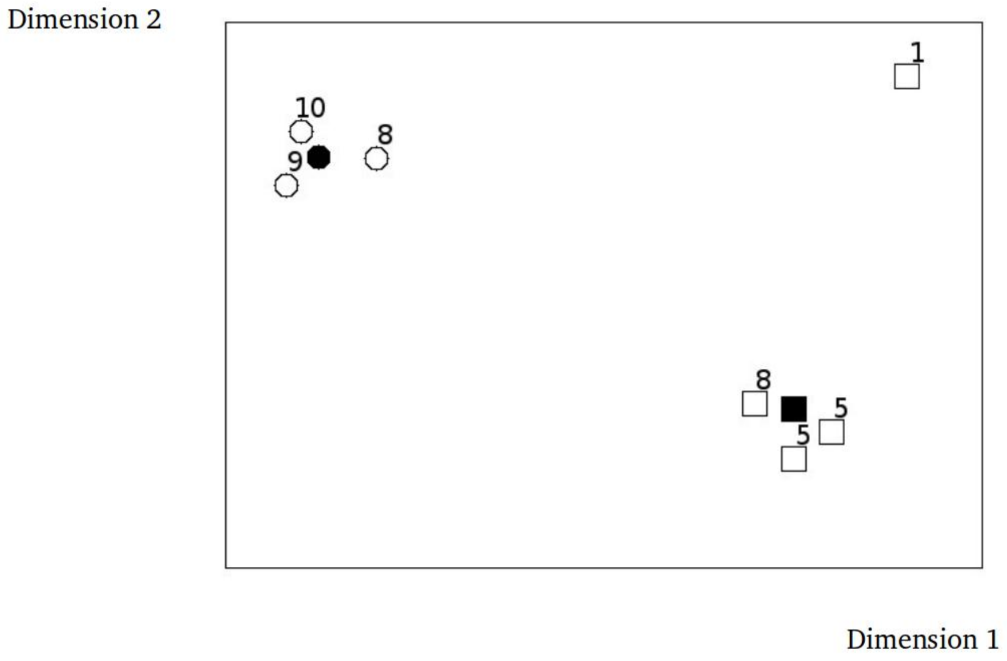


Figure 6.3: Routing model with clustered query-confidence pairs using weighted K-means clustering.

For each cluster c, we can develop now a function f_c for confidence estimation. The confidence estimation for a query is always based on a cluster c. If there are multiple clusters in the routing model, we will have to perform multiple estimations, one for each cluster and take the maximum. The function f_c takes the euclidean distance of the query and the weighted centroid of cluster c as input and returns the estimated confidence of the query.

We specify the following requirements on the function f_c:

Requirement 1
If the distance from the query $\mathbf{q}$ *to the weighted centroid* $\bar{c}_{weighted}$ *is zero, the function should return the weighted average confidence score of cluster* c*.*

$$f_c(0) = \mu_{score} \qquad \mu_{score} = \frac{\sum_{(\mathbf{q},C)\in c} \frac{C}{|c|} C}{|c|}.$$

Requirement 2
The function f_c should decrease with increasing distance between query and weighted centroid of the cluster. For an increasing distance, the score should go against zero. To be more precise, we define the function value as follows:

$$f_c(\sigma_c) = \mu_{score} - \sigma_{score}.$$

The standard deviation of confidence values σ_{score} of queries in the model is defined as:

$$\sigma_{score} = \frac{\sum_{(\mathbf{q},C)\in c}(C - \frac{\sum_{(\mathbf{q},C)\in c} C}{|c|})}{|c|}.$$

Requirement 3
We define f_c to be a standard gaussian function:

$$f_c(x) = a_1 exp(\frac{-x^2}{2a_2^2})$$

with constants a_1 and a_2.

Theorem 1
With these requirements, we get the following function for confidence estimation in the routing models:

$$f_c(x) = \mu_{score}(\frac{\mu_{score} - \sigma_{score}}{\mu_{score}})^{\frac{x^2}{\sigma_c^2}}$$

Proof 1
Insertion of requirements 1 and 2 in requirements 3 and transformation gives:

$$a_1 = \mu_{score} \qquad a_2^2 = \frac{-\sigma_c^2}{2\ln(\frac{\mu_{score}-\sigma_{score}}{\mu_{score}})}$$

Insertion of constants a_1 and a_2 in requirement 3 and transformation gives Theorem 1.

With Theorem 1, we can define the confidence estimation CONFIDENCEESTIMATION($\mathbf{q}$) of a routing model RM as:

$$\text{CONFIDENCEESTIMATION}(\mathbf{q}) = max_{c\in RM}(f_c(||\bar{c}_{weighted} - \mathbf{q}||))$$

6.1.3 Maintenance of routing models

Initially, the routing models on a node contain no information. In order to fill the routing models with information, nodes regularly issue **exploration queries** to neighbours. Neighbours answer exploration queries with a **feedback** value. Nodes use the feedback values in order to **update their routing models**. We introduce these three concepts in the following.

In order to learn about the neighbours expertise areas, nodes regularly issue **exploration queries** to some neighbours. These exploration queries are conceptually different to normal queries (sometimes called *exploitation queries*), their only purpose is to gain routing information for the routing models.

A node p_1 first decides on a point in the space for that it would like to have more information about how good neighbours think, they can answer queries at this point. In our implementation, we have chosen a random point in the space as exploration query. Then, node p_1 sends the exploration query to a set of neighbours. It chooses this set according to a selectivity parameter s. Selectivity specifies the number of neighbours to which the query has to be forwarded. For example, if selectivity equals two, it will be sent to two randomly selected neighbours.

The exploration query is associated with another parameter, the hop count h. This parameter specifies the depth of the exploration query. When a node receives an exploration query, it will forward the same exploration query to its neighbours, until the maximal hop count is reached.

We can see an example of these two concepts in Figure 6.4. A node initiates an exploration query with $h = 2$ and $s = 2$. Each node on the way chooses two neighbours and forwards the exploration query to those. When the maximal depth of the exploration query is reached, the forwarding will be stopped.

Each node that receives an exploration query gives a **feedback** to the last forwarder of the exploration query that can be used as routing information. The feedback is basically an estimation of the confidence value that would be retrieved for a normal (exploitation) query at this point in the space. This idea is based on the reinforcement approach, we have presented in Chapter 2. The hope is, that the estimations will become better over time, when each node learns about the estimations of its neighbours.

We have identified two different feedback mechanisms: *immediate feedback* and *delayed feedback*. Using immediate feedback, a node that receives an exploration query determines the maximal value of the local confidence for the exploration query and the estimated confidence based on its local routing models. Before forwarding the exploration query to any other node (if hop count is specified accordingly), it returns this feedback value to the last sender of the exploration query. Hence, immediate feedback is based only on the local information of a node. For an example of immediate feedback see Figure 6.5.

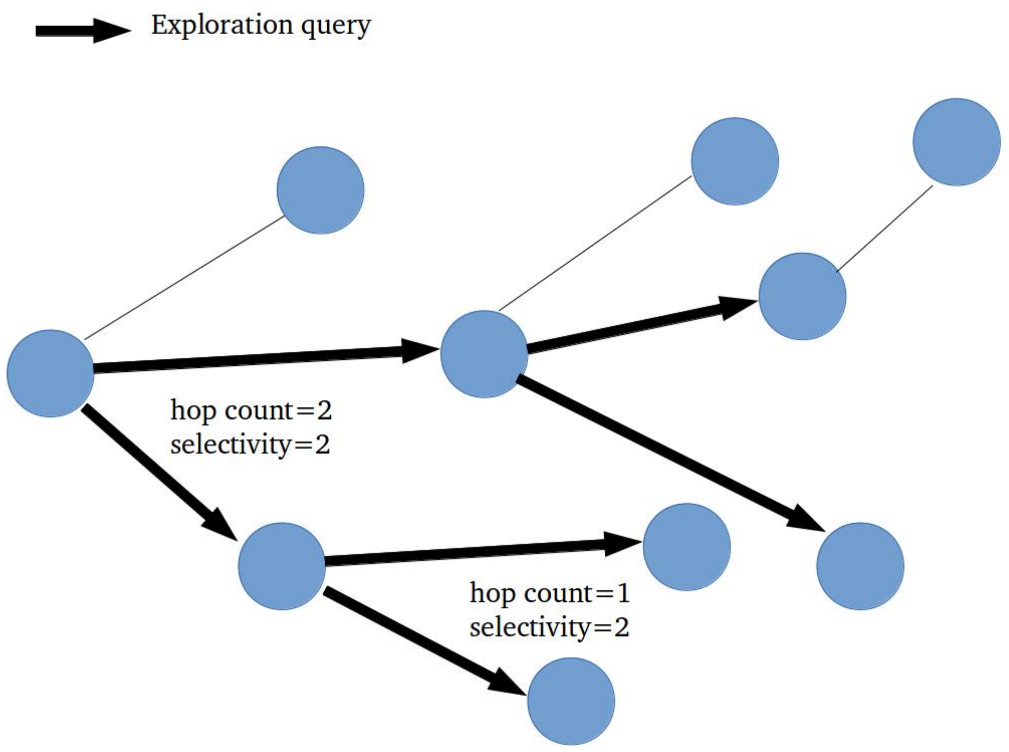


Figure 6.4: Paths of an exploration query with hop count 2 and selectivity 2.

The other mechanism is delayed feedback (see Figure 6.6). On receiving an exploration query, a node forwards the exploration query according to selectivity and hop count to its neighbours. Once it has received feedback for the exploration query from all of these neighbours, it gives feedback to the last sender. In this case, the feedback value is the maximum of the local confidence and each neighbours feedback. If there is no neighbour or maximum hop count is reached, the feedback is, as above, only based on its local information in the data and routing models.

In order to **update the routing models** with query-confidence pairs, nodes issue exploration queries and receive feedbacks from neighbours, as we have seen. For each exploration query $\mathbf{q}$, a node selects the neighbour from which it has received the highest feedback value $\hat{C}$ and updates the routing model of this neighbour with the query-confidence pair $(\mathbf{q}, \hat{C})$.

In dynamic environments, query-confidence pairs can become stale and should be removed from the routing models. We have identified two concepts to solve this problem. In order to store only fresh query-confidence pairs, a node can maintain only a window of query-confidence pairs of size w. The routing model simply forgets older queries.

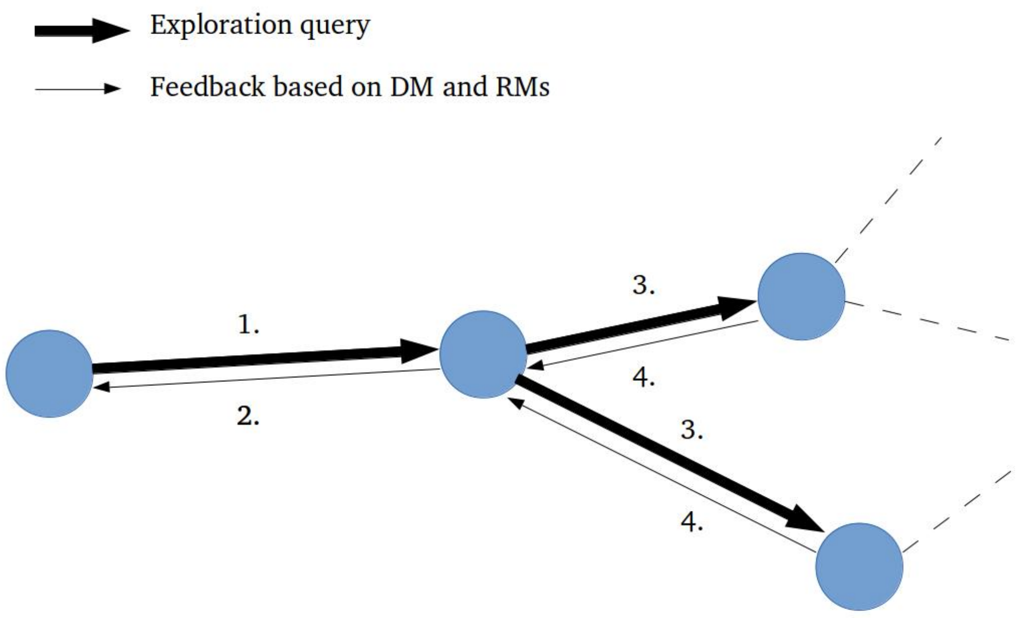


Figure 6.5: Example of immediate feedback.

Another technique to handle this problem is to smoothly decrease the meaning of older queries in the routing model by multiplying regularly a decay factor α to the confidence values of the stored queries (compare [DCCC05]). When the confidence part of a query-confidence pair falls beyond a certain threshold Θ, it will be removed from the routing model. With this method, query-confidence pairs become unimportant over time with respect to the confidence estimation (see weighted k-means clustering above).

In the following, we present how the protocol could be realized in pseudocode.

6.2 Protocol Description

This section is structured like the protocol overview: First, we recap the protocol specific parameters and notation. This is followed by the basic routing algorithm. Afterwards we give algorithms for exploration. We end this chapter with a short investigation of the implementation of a routing model.

We have the following system parameters:

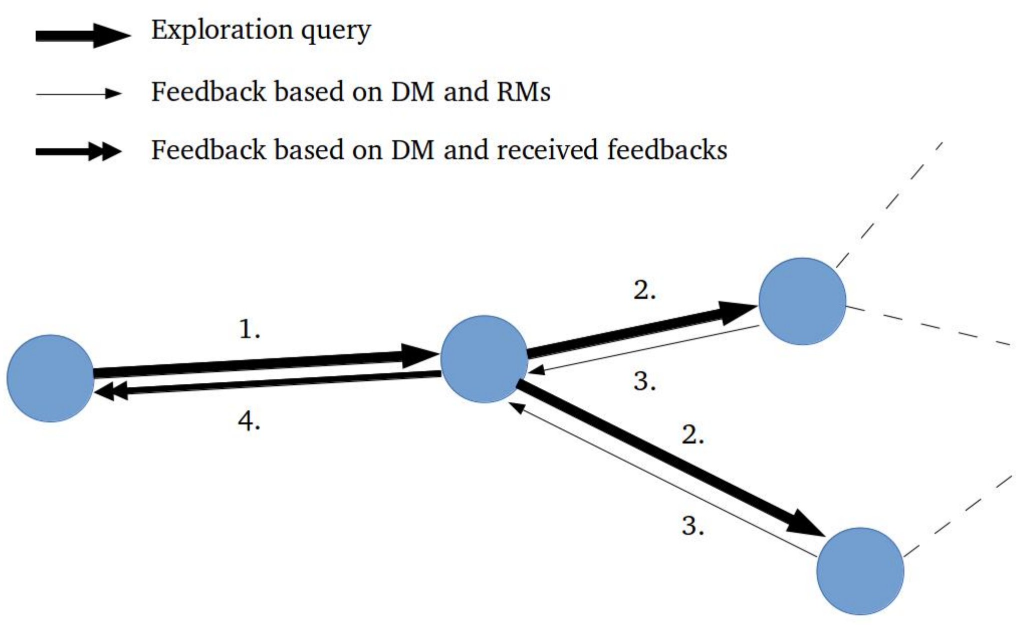

Exploration query
Feedback based on DM and RMs
Feedback based on DM and received feedbacks
1.
4.
2.
3.
2.
3.

Figure 6.6: Example of delayed feedback.

- The selectivity parameter $s \in [0, 1]$ specifies the fraction of neighbours to which a node forwards an exploration query
- Each exploration query has a maximal depth with respect to the initial issuer. This maximum hop count is specified by $h \in \mathbb{N}$.
- In the local routing models, a node can store query-confidence pairs. The maximal number of such pairs is given by the window size w. Regularly a node will decay each confidence value of the stored pairs by the specified decay factor α. Hence for a pair $(\mathbf{q}, C)$ the confidence value will be decayed like $C = \alpha C$. It may occur that some confidence values fall below a certain threshold Θ. Then the query-confidence pair will be removed from the routing model.

In addition to the protocol specific parameters, each node maintains some variables to run the protocol correctly:

- A set of adjacent nodes, called $neighbours$.

- The $dataModel$ can be seen as module that manages the local data and offers a possibility to calculate the confidence score for each cluster. As in the previous section, we define the best local cluster to be $\hat{c}$. The response vector $\hat{\mathbf{v}}$ will be the best local clusters centroid:

$$\hat{c} = argmax_{c \in dataModel} C(c, \mathbf{q}) \qquad \hat{\mathbf{v}} = \frac{\sum_{\mathbf{v} \in \hat{c}} \mathbf{v}}{t_{\hat{c}}}$$

- A set of routing models $routingModels$, one routing model for each neighbour. The routing model differs from the DA approach. Each model maintains query-confidence pairs instead of data vectors. The idea is to learn expert areas of the neighbours with the help of exploration queries.

In Algorithm 6.1 we show the behaviour of node p_i on receiving a query from node p_j. Node p_i will answer the query q locally with the best local data $\hat{\mathbf{v}}$ (line 3), if the local data model possesses data that has high enough confidence or if there is no other node to forward the query to. Otherwise, it will forward the query to the neighbour, whose routing model gives the highest estimated confidence for the query. This neighbour is determined with $bestNeighbour(\mathbf{q})$ in line 6.

Algorithm 6.1 Node p_i receives query message

```
1: on $receive(QUERY, q = (\mathbf{q}, C^{\mathbf{q}}_{min}), visited)$ from node $p_j$ do :
2:     if $(C^{\mathbf{q}}_{min} < C_i(\mathbf{q})) \vee (neighbours \setminus \{p_j\} = \emptyset)$ then
3:         $send(RESPONSE, \hat{\mathbf{v}}, C_i(\mathbf{q}), q, visited)$ to node $p_j$
4:     else
5:         $visited \leftarrow visited + [p_i]$
6:         $send(QUERY, q, visited)$ to $bestNeighbour(\mathbf{q})$
7:     end if
8: endOn
```

When a node decides to answer the query locally, the response will be send backwards along the path of the query. When a node receives a response (see Algorithm 6.2) and is able to answer the query locally with a higher confidence value, it will replace the received response with its own and adapt the confidence value for the next receiver (line 4). Otherwise it sends the response backwards without any change.

Regularly, each node should explore its environment by sending an exploration query $\mathbf{q}$ to a number of neighbours according to the selectivity parameter s (Algorithm 6.3, line 2). The depth of this exploration query is given by h.

Also on a periodical basis, each node decays the query-confidence pairs in its local routing models (line 6). With this, old queries were *smoothed* out of the routing models.

After receiving an exploration query (line 11), a node will check, whether the maximal number of hops is reached. If so, the node will have to determine the feedback for the sender of the

Algorithm 6.2 Node p_i receives response message for a query

```
1: on $receive(RESPONSE, R, C, q, visited)$ from node $p_j$ do :
2:     $visited.removeLast()$
3:     if $C_i(\mathbf{q}) > C$ then
4:         $send(RESPONSE, \hat{\mathbf{v}}, C_i(\mathbf{q}), q, visited)$ to $visited.last()$
5:     else
6:         $send(RESPONSE, R, C, q, visited)$ to $visited.last()$
7:     end if
8: endOn
```

exploration query. For this, it takes the maximum of the local confidence value and the routing models estimations of the confidence values for the query (line 13).

If the maximal number of hops is not reached yet, the node randomly selects a subset of its neighbours according to the selectivity parameter s (line 15). The node forwards the exploration query to these neighbours and waits for all the feedback responses. It returns the best feedback value from all neighbours to the issuer of the exploration query (line 19) or the local confidence, if it is better than each feedback.

Now, the node updates the best neighbours routing model with the exploration query and the feedback value of this neighbour (line 20). The underlying assumption is, that if this neighbour thinks, it can answer the query best, future queries that are in this area should be forwarded more likely to this one.

We will give now a possible implementation of the routing model (see Algorithm 6.4). We thought of it as a module that offers some functionality like determining an estimation of confidence for a query (line 10), which is basically the maximal confidence estimation over all clusters in the routing model. We have discussed the method of estimating the confidence score above. In line 2, we have given the update function for new query confidence values. Only query-confidence values will be stored that are better than the threshold value Θ. If there is a maximal window size, the oldest pairs will be removed from the routing model (line 5).

For estimating confidence values (line 10), we first need to cluster the queries with the method in line 13. Queries that have a higher confidence score, should have more impact on clustering. This can be done by performing weighted k-means with the confidence scores as weights. Overweighting these queries has the effect of slowly forget about old or bad queries that are still in the model.

The weighted k-means algorithm requires the number of clusters as input. To make the protocol more flexible and independent on system parameters, we decided to estimate this parameter using G-means (line 14).

Algorithm 6.3 Node p_i: exploration query

```
1: set $received$
2: **on** $explorationTimeout()$ **do** :
3:     $\mathbf{q} \leftarrow getRandomQuery()$
4:     $send(EXPLORE, \mathbf{q}, h)$ to node $p_i$
5: **endOn**
6: **on** $decayTimeout()$ **do** :
7:     **for** $rm \in routingModels$ **do**
8:         $rm.\textsc{decay}()$
9:     **end for**
10: **endOn**
11: **on** $receive(EXPLORE, \mathbf{q}, h)$ from node $p_j$ **do** :
12:     **if** $h = 0$ **then**
13:         $send(FEEDBACK, \mathbf{q}, max(C_i(\mathbf{q}), \textsc{confidenceEstimation}(\mathbf{q})))$ to $p_j$
14:     **else**
15:         $neighbours' \leftarrow selectNeighbours(neighbours, s)$
16:         $received = \{\}$
17:         $send(EXPLORE, \mathbf{q}, h-1)$ to $neighbours'$
18:         $waitForFeedbacks(neighbours')$
19:         $send(FEEDBACK, \mathbf{q}, max(getBest(received), C_i(\mathbf{q})))$ to $p_j$
20:         $getBestRoutingModel(received).\textsc{update}(\mathbf{q}, getBest(received))$
21:     **end if**
22: **endOn**
23: **on** $receive(FEEDBACK, \mathbf{q}, C)$ from node $p_j$ **do** :
24:     $received \cup (\mathbf{q}, C, p_j)$
25: **endOn**
```

In line 17, we give the decay function, that decays each confidence value by the specified decay factor α and removes query-confidence pairs in the routing model, that fall below the threshold Θ.

Until now, we have assumed that the dimensionality of the query and the routing and data models are the same. But as we have already mentioned in Chapter 3, the dimensionality can be different. In the following, we show how we can handle different dimensionality of query and data and routing models.

6.3 Dimensionality Reduction

The curse of dimensionality [VF05] is a phenomenon that arises, when dealing with high dimensionality. It states, that properties of the data set, like the clustering of the data, vanish

Algorithm 6.4 Routing model

```
list $queryConfidencePairs$
procedure UPDATE($\mathbf{q}, C$)
    if $C >= \Theta$ then
        if $|queryConfidencePairs| >= w$ then
            $queryConfidencePairs.removeFirst()$
        end if
        $queryConfidencePairs$.APPEND(($\mathbf{q}, C$))
    end if
end procedure
function CONFIDENCEESTIMATION($\mathbf{q}$)
    Please see the last section for how to compute the estimation
end function
function CLUSTER($V$)
    $goodK \leftarrow |GMeans.cluster(queryConfidencePairs)|$
    return $WeightedKmeans.cluster(queryConfidencePairs, goodK)$
end function
procedure DECAY( )
    for $(\mathbf{q}, C) \in queryConfidencePairs$ do
        $C \leftarrow C * \alpha$
        if $C < \Theta$ then
            $queryConfidencePairs.remove(\mathbf{q}, C)$
        end if
    end for
end procedure
```

in higher dimensions. In Chapter 4, we give a method of how to compute the confidence score for a query. This method is based on data clusters. The implicit assumption here is, that the data can be clustered at all. Because of that, it makes sense to reduce some dimensions of the local data model, so that data clusters become better. We have to come up with a method that selects dimensions that can be reduced.

In the routing model, we compute an estimation of confidence based on query clusters (see previous sections). This computation requires also good clusters. Like in the local data model, it is a good idea to reduce dimensionality, so that the clustering property of the data is preserved. In the following, we show a method to reduce both models, the local routing model and the local data model.

The goal is to get a lower dimensional space, where the clustering of the data is better as in the high dimensional space. We strive to achieve this property by i)clustering the data separately for each dimension with G-means clustering (see [HE03]), ii) determining the average variance of a cluster for each dimension and iii) reducing the dimensions with highest average variance of a cluster. The idea behind this method is that clusters will be interpreted as good, if their variance is low (compare Chapter 4).

In Figure 6.7 we have given an example of the variance based reduction. We have plotted three-dimensional data, where there are two clusters in the space. Dimension 1 disturbs the clustering, as data is spread widely in this dimension. The above method clusters each dimension separately and determines for each dimension the average variance of a cluster. Obviously, the average variance of clusters in dimension 1 (there is only one cluster) has the highest variance and should be reduced. In Subfigure 6.7e, we can see the reduced space, where there are very well separated clusters.

With this method, we can reduce dimensionality in the local models. It is possible that the query has different dimensions than the model. In this case, we take the intersection of the dimensions of the model and the query as basis to compute the confidence or estimated confidence score. If the intersection of dimensions is the empty set, we will have no confidence in the data for the query and should return zero as confidence or confidence estimation.

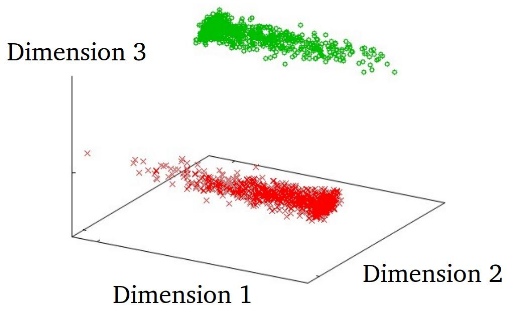
We end this chapter with a short discussion of the approach.

6.4 Discussion

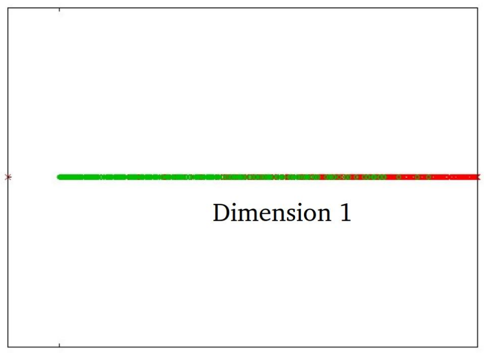
In contrast to the data aggregation (DA) routing approach, the query learning (QL) routing approach uses previous queries and their feedbacks to fill the routing models with information. The updating can be performed during the protocol execution. There is no direct dependency between a node and a distant participant of the protocol. A node needs only feedback from its direct neighbours to learn about reachable data. This makes the approach more robust against node failures.

We have given a method for extracting an estimation of confidence out of the routing models, even in the presence of diverse dimensionality of the query and the model. Additionally, we have shown how dimensionality reduction can be integrated into this approach. Note, that methods for dimensionality reduction are well-studied in research [YL03, PHL04]. Other dimensionality reduction techniques could be integrated easily into the approach. We just need a method that determines a lower dimensional space with good clustering properties.

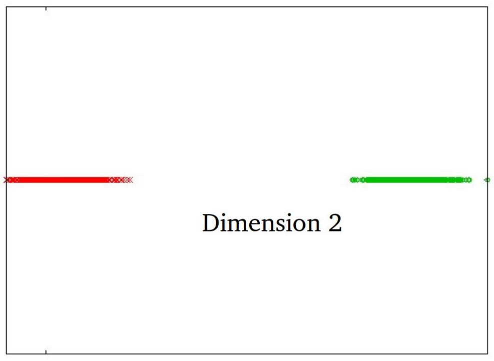
In the next section, we give an evaluation of both approaches.



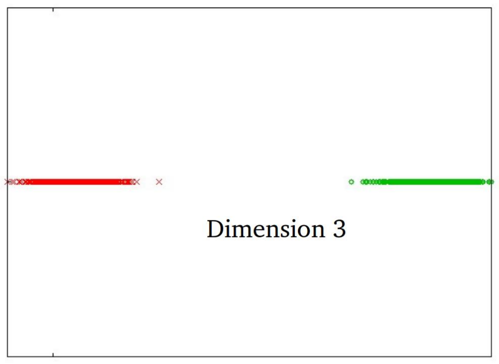
(a) Data in three-dimensional space.



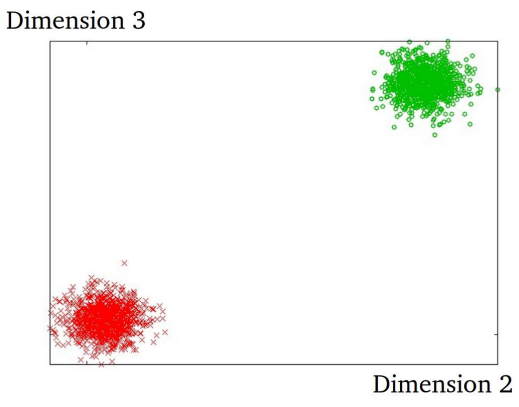
(b) Data projected to dimension 1.



(c) Data projected to dimension 2.



(d) Data projected to dimension 3.



(e) Data in reduced space (dimensions 2 and 3).

Figure 6.7: Dimensionality reduction example.

7 Evaluation

In this chapter, we evaluate the *query learning (QL) routing approach* and the *data aggregation (DA) routing approach* with respect to the **accuracy** of the retrieved confidence scores.

Definition 1
***Accuracy** measures the ability of the system to retrieve high confidence values for a query* $\mathbf{q}$. *Accuracy of one means, that the system always finds the highest confidence score in one of the local data models. We calculate accuracy as the confidence value* $C(\mathbf{q})$ *that was retrieved by the system, divided by the highest possible confidence value in any local data model given by* $\hat{C}(\mathbf{q}) = max_{i \in \{1,\ldots,n\}} C_i(\mathbf{q})$.

We have used the network simulator "PeerSim" [MJ09] to simulate and evaluate our routing approaches. Using a simulator does not influence the accuracy of the system.

This chapter is structured as follows. Data can be localized in the system or not localized. When data is localized, nodes that are topographically close are also semantically close (i.e., neighbours have approximately the same data in their data models). In the next section, we present our topology generator that generates a localized simulation network. Afterwards we evaluate the accuracy of the QL routing approach for different parameter settings. At the end, we compare the presented routing approaches and give the accuracy of the QL approach in presence of dimensionality reduction.

7.1 Topology

We investigate our approaches with respect to localized topologies. For our tests, we have implemented a topology generator that generates acyclic, undirected and localized simulation networks for our experiments. In practice, topologies with localized data could arise for different reasons, e.g., because neighbours observe the same environment in sensor networks.

An example of a generated, small topology with 100 nodes is given in Figure 7.1. Each point (seed vector) represents one node of the network. The topology generator initializes the data models of the nodes with data that is distributed around these seed vectors.

The construction of the network is performed hierarchically and starts with the seed of the source node and a circle with a certain radius around the seed. Each such cycle represents the maximal seed distribution of the children. The source node chooses some seeds in its circle and builds for each seed one direct child. The children iterate this process recursively with a smaller radius. With this, we have a method for constructing a tree topology with localized data.

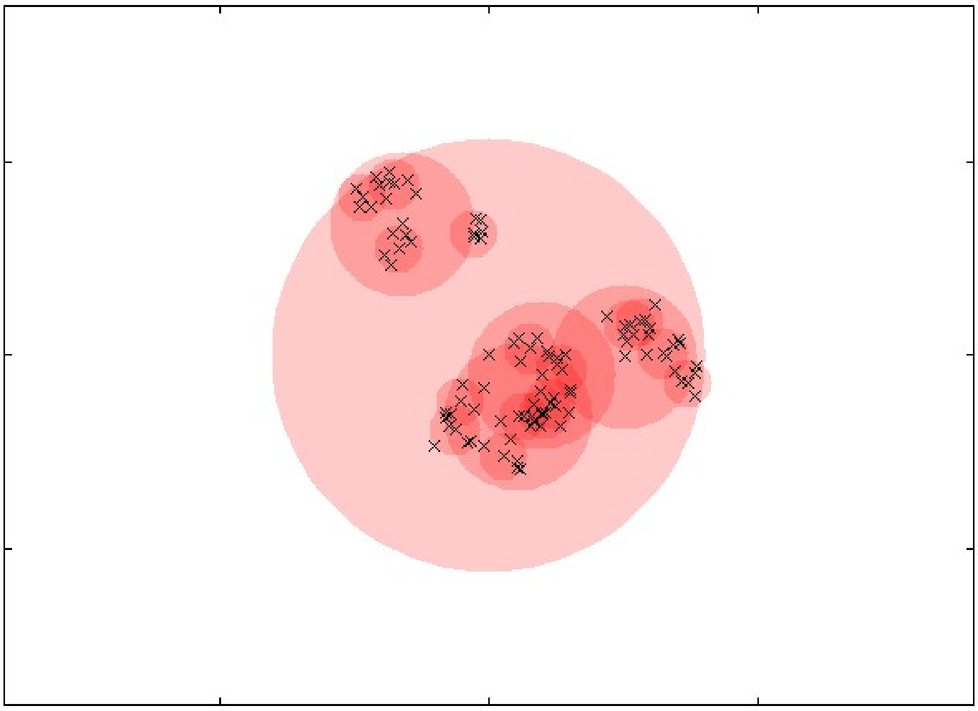


Figure 7.1: Localization of data. Nodes are associated with points in the vector space.

The generated topology for 100 nodes is shown in Figure 7.2. The graph is undirected and acyclic.

All experiments are performed on a topology that was generated with the presented method. In the next section, we evaluate the QL routing approach in order to find appropriate protocol parameters.

7.2 Parameters for the QL Approach

There are two different kind of queries (see Chapter 6): *exploration queries* that nodes can send to their neighbours in order to retrieve routing information and *exploitation queries* or simply *queries* that were issued by an application of the system and were routed according to

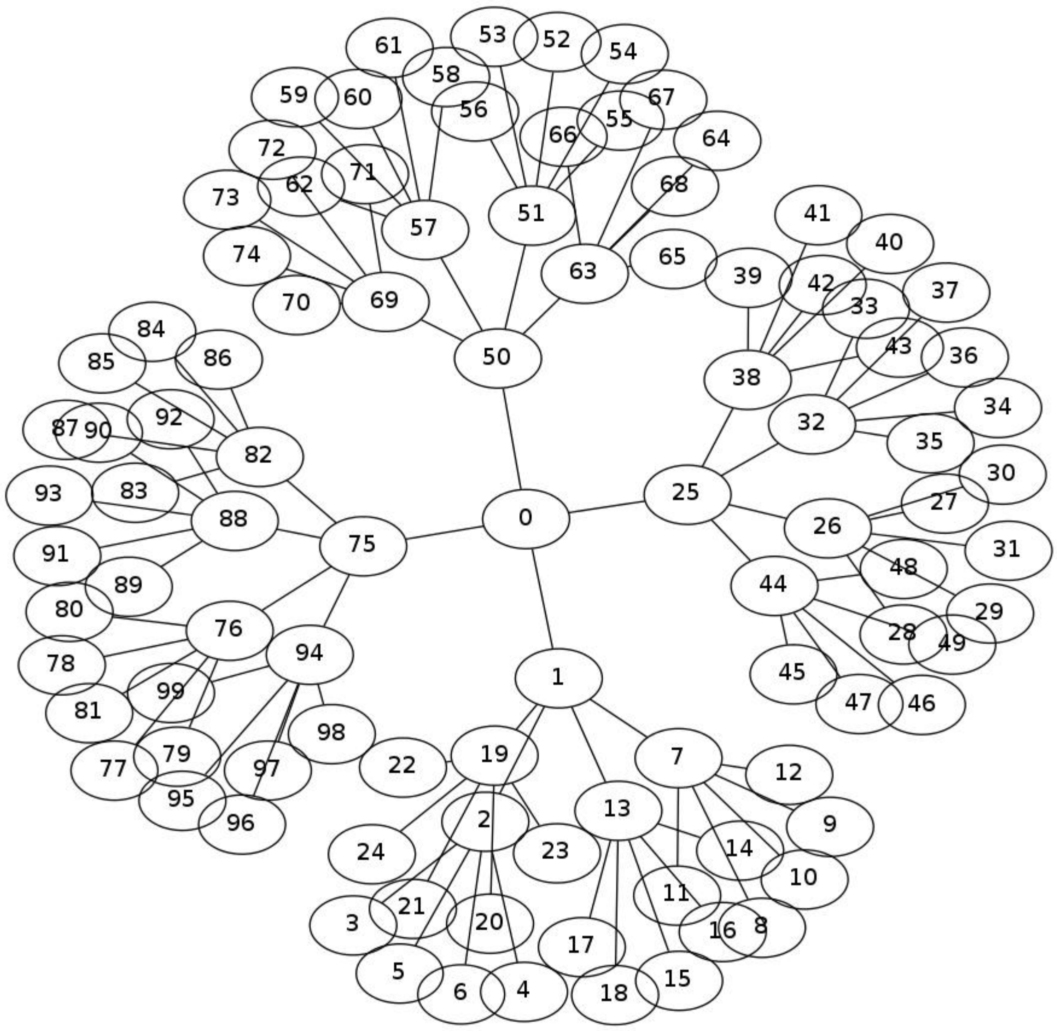


Figure 7.2: Topology with 100 nodes

the information in the routing models. For all experiments, (exploitation) queries were routed to the best neighbour according to the routing models. For the determination of the accuracy of the system, we use only the exploitation queries.

The information for the routing models were retrieved by exploration queries. There are two parameters for exploration queries: *selectivity* s, i.e., the number of neighbours to send the exploration query, and *hop count* h, that is the depth of an exploration query. With increasing selectivity or hop count, the number of nodes that were affected by a single exploration query increases, too. We are interested in a parameter configuration that results in high accuracy, while the system is still scalable.

In Figure 7.3 we have plotted the average accuracy of the QL routing approach for different parameter configurations. The horizontal axis represents the selectivity value for exploration queries. For each selectivity value, we have measured accuracy for three different hop count values: one, three and infinity. For the experiments, we have used different localized topologies with 1000 nodes. Each node has maximal five neighbours. We have averaged the accuracy of the retrieved confidence values. As we can see, with increasing selectivity and hop count, the accuracy of the approach increases as more information can be gained by one exploration query round. This is basically a trade-off between accuracy and scalability. For instance, flooding the network with exploration queries results in poor scalability but high accuracy (see Figure 7.3).

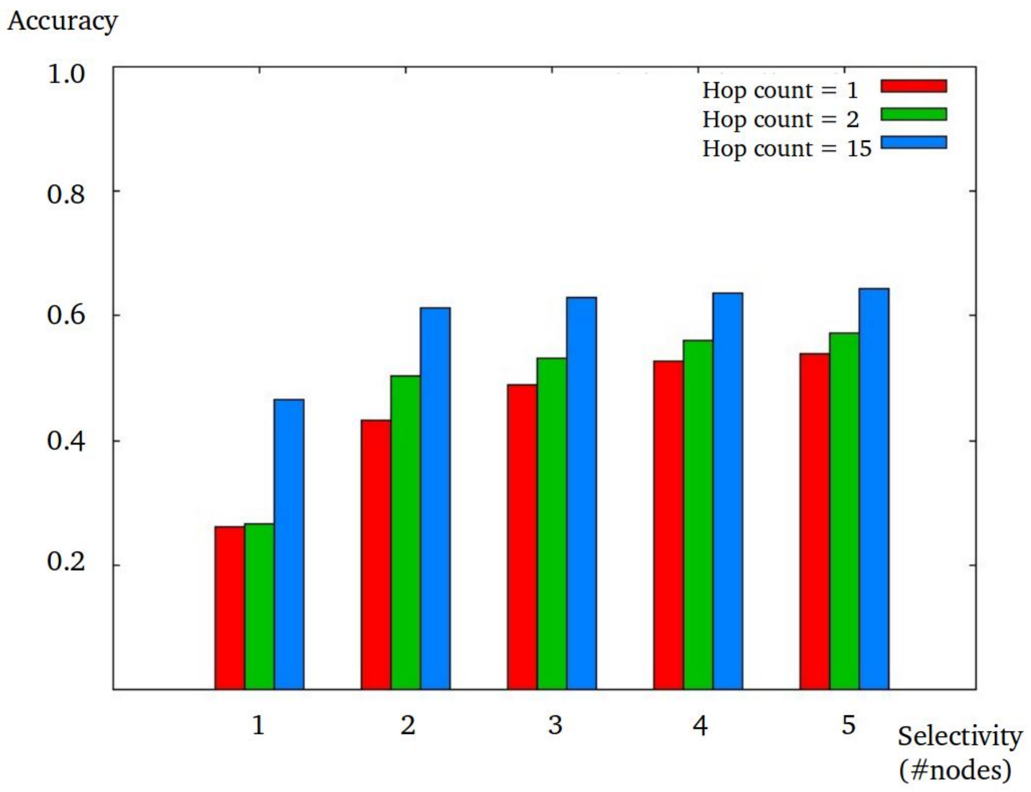


Figure 7.3: Accuracy of QL routing for different selectivity and hop count parameter values.

For further experiments, we set selectivity $s = 5$ and hop count $h = 2$, because the accuracy is still high and the exploration queries have only local scope. In the following, we compare accuracy for different routing approaches.

7.3 Comparison Routing Approaches

In Figure 7.4, we compare the QL and DA routing against a random routing approach (Rand). In the random routing approach, the query is forwarded into a random direction. Therefore, it can be seen as a lower bound for accuracy. Because of scalability issues, we have modified the DA approach slightly: when receiving cluster information, a node regenerates only half of the data vectors per cluster. This has a slight negative influence on the accuracy, but enables the simulator to run a high number of nodes.

The DA approach has better accuracy for big topologies (1000 and 10000 nodes) than the other routing approaches. However, the QL routing approach is very close to it. Both approaches outperform very easily the random-based routing due to the additional information in the routing models.

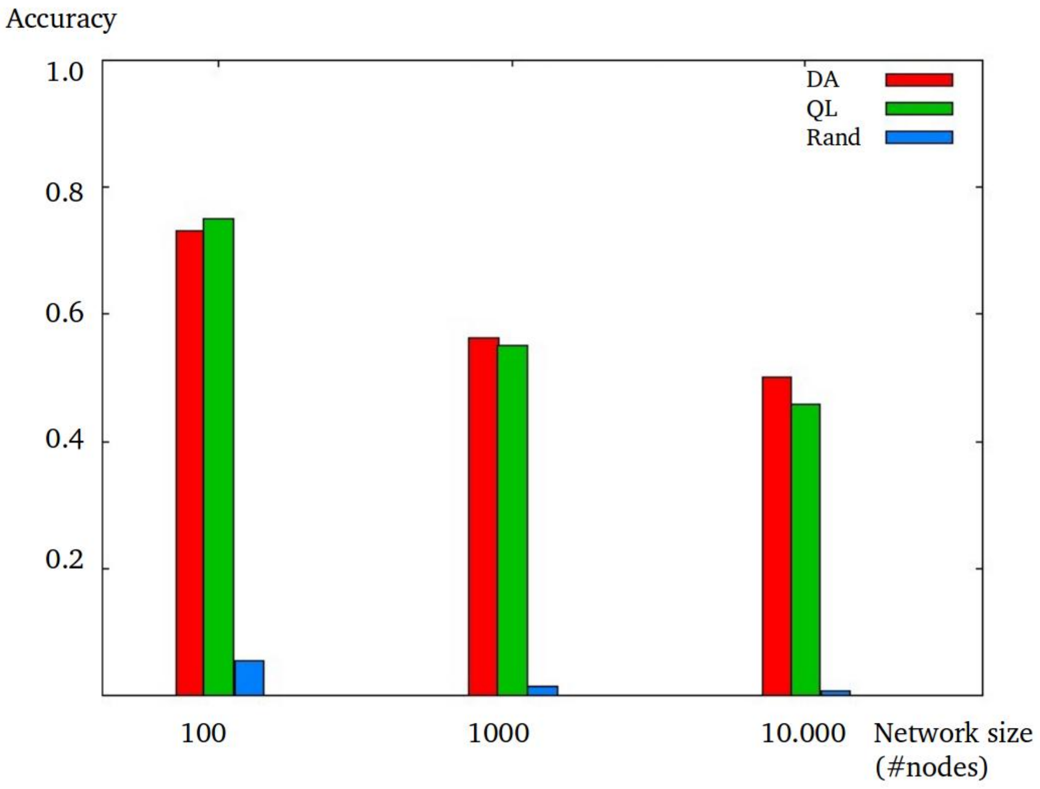


Figure 7.4: Comparison of accuracy between data aggregation (DA), query learning (QL) and random (Rand) routing for different network sizes.

The protocol execution is organized in *cycles*. In each cycle, each node may perform some protocol specific steps like exploration of the network. Additionally, we issue in each cycle 100 random queries to random nodes in the system in order to determine the current accuracy of the system.

Initially, the routing models of both approaches contain no information. During the protocol execution, nodes explore the network and fill their routing models with routing information. The DA approach exchanges compact clustering information. In Figure 7.5, we can see the learning phase of the DA approach. We have plotted the accuracy of DA routing for a small topology (100 nodes). The horizontal axis is the number of cycles that were performed. The accuracy of a cycle is the average accuracy of all queries in that cycle. Initially, the accuracy is not higher than random routing. Then, nodes start to exchange compact clustering information and fill their models with data. After the exchange is ready, all nodes have information about their neighbours and accuracy increases.

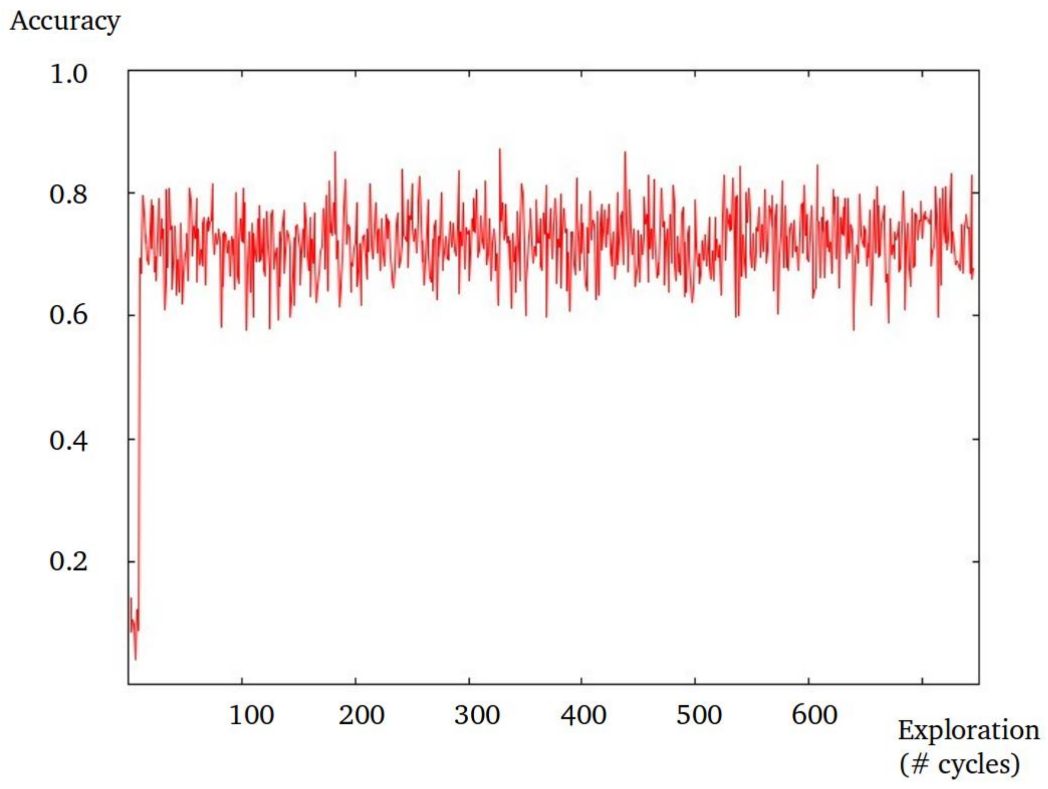


Figure 7.5: Learning in the DA approach.

In contrast to the DA approach, learning routing information in the QL approach is done in smaller steps (see Figure 7.6). Here, in each cycle, each node initiates one exploration query. After approximately 500 cycles, the system reaches its maximal accuracy level for this configuration.

In comparison, the DA approach shows better accuracy than the QL approach for big topologies. The benefits of the QL approach are in the finer granularity of learning. The QL approach converges smoothly to its maximal accuracy level, while the DA approach has either very low accuracy or very high. There is nothing in-between.

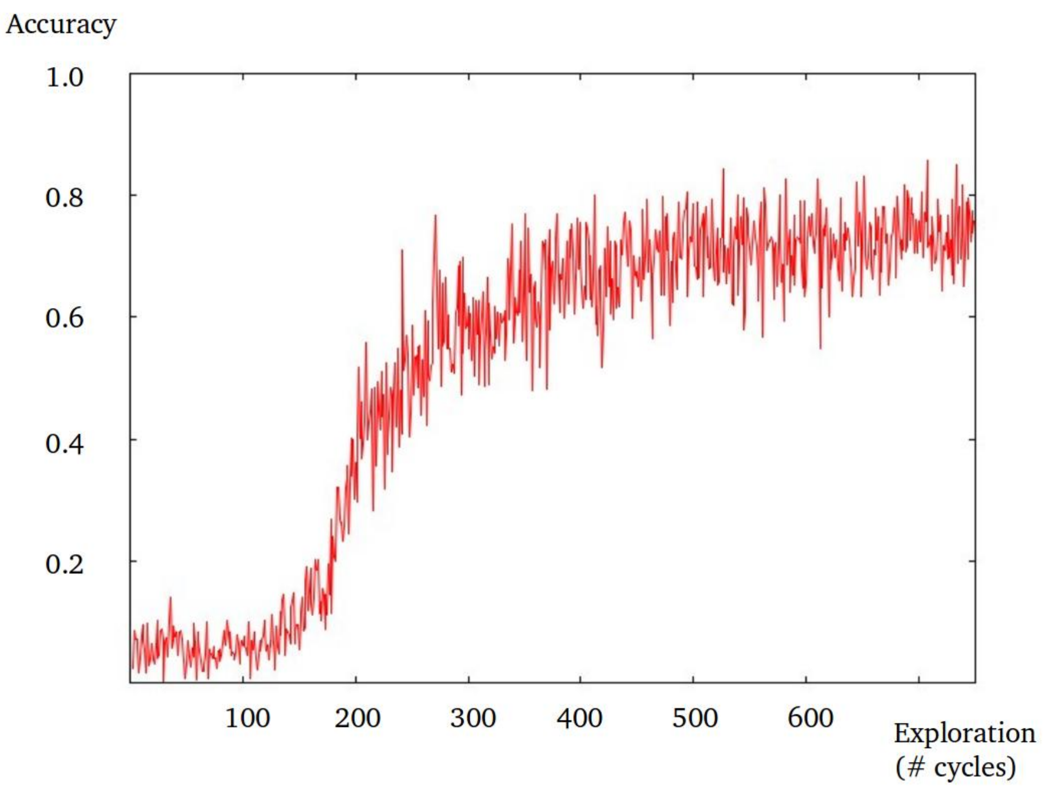


Figure 7.6: Learning in the QL approach.

7.4 Dimension Reduction

We have implemented dimensionality reduction in the QL routing approach (compare Chapter 6). The evaluation in this section is based on a localized topology with 1000 nodes. Each node is initialized with 10-dimensional data (instead of two-dimensional as previously). The data on each node has in 7 dimensions a high variance and in 3 dimensions low variance. The dimensions with lower variance are called *unimportant dimensions*. For different nodes, different dimensions are unimportant. But the unimportant dimensions are also localized in the network, i.e., that neighbours consider approximately the same dimensions as unimportant.

Nodes perform dimensionality reduction of the local data in order to overcome the curse of dimensionality. Then, random 10-dimensional queries were issued to random nodes. In Figure 7.7, we compare an approach, where nodes reduce the dimensionality of their routing models against an approach, where nodes do not perform such a reduction in the routing models. As we can see, the accuracy is higher, when dimensionality is reduced in the routing models. The explanation is that the (reduced) information in the routing models fits better to the (reduced) data in the data models of the nodes.

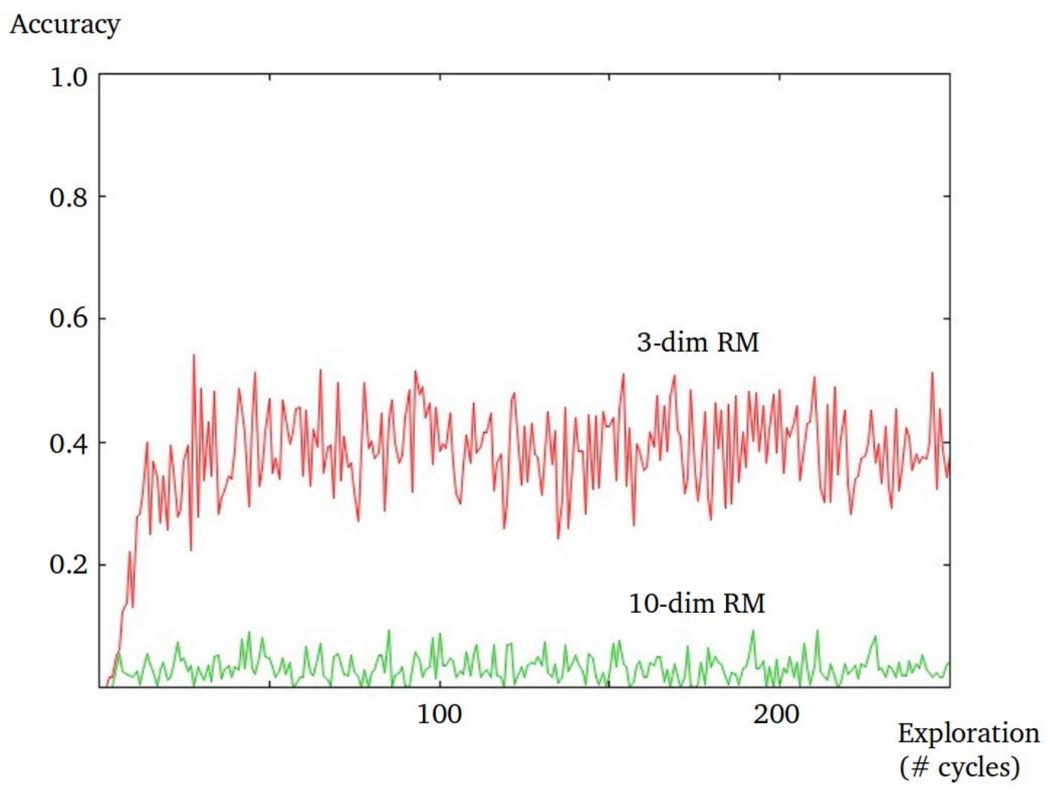


Figure 7.7: Effect of dimensionality reduction in the routing models (RM).

In essence, when nodes have to deal with high-dimensional data and therefore decide to reduce dimensionality in their local data models, it has a positive effect on the accuracy to reduce dimensionality in the routing models. The reduction is performed only with local knowledge in the routing models.

We end this thesis with a conclusion and give some future work that can be done.

8 Conclusion and Future Work

The main goal of this thesis was to design and implement a distributed system that stores multidimensional data and can be queried in order to retrieve confidence values for the queries. Therefore, we have formalized the idea of data confidence, so that the system is able to determine how much knowledge it has about a query, even if it has never seen a data vector that is similar to the query.

A main challenge for the system was the heterogeneous and dynamic environment in which a query should be routed to a node that has high confidence in it. We have designed the protocols to be scalable in terms of message overhead and routing information: nodes use only local communication and build a model of their environment instead of maintaining exact routing tables.

The problem of high dimensionality of data on different nodes was solved with a method for dimensionality reduction. In the reduced space, previously hidden clusters become more visible. This is crucial for our confidence measure. Therefore dimensionality reduction is an important tool for nodes to determine their confidence in a query. Also, we have shown in the evaluation how dimensionality reduction can be useful for routing, too.

The different techniques in our approach, like dimensionality reduction, a measure for confidence or how to use the routing models to determine the next hop for a query, can be seen as modules that are in fact interchangeable with possible alternatives. For example we decided for a relatively simple dimensionality reduction technique, but we can imagine to use more sophisticated techniques that can be used instead, perhaps at the cost of more resource consumption on the nodes.

In our evaluation, we have investigated the accuracy of the routing approaches. We have left it for future work to evaluate the system against dynamic changes of the topology and the data on the nodes. Furthermore, the system could be extended to handle data insertions or delete operations, perhaps to provide some localization of data by design. An argument to left this open was, that data is often inherently localized, because neighbours often perceive similar observations of their environment.

A limitation of the confidence measure is in the meaning of a data vector. We have assumed, that a high number of data results in a high confidence score. While this is reasonable in general, we can also imagine data vectors that are of different implicit quality. Then, the implicit quality of the data vectors in a cluster should also have some influence on the confidence measure.

We can even think of data vectors that are “negative observations” and that should make the confidence score worse for similar queries. However, this is also left open for future work.

We strongly believe, that our system for confidence retrieval can be the basis for interesting applications that make use of the provided quality information about the data.

Bibliography

[ABGS+14] V. Andrikopoulos, A. Bucchiarone, S. Gómez Sáez, D. Karastoyanova, C. Mezzina. Towards Modeling and Execution of Collective Adaptive Systems. In A. Lomuscio, S. Nepal, F. Patrizi, B. Benatallah, I. Brandić, editors, *Service-Oriented Computing – ICSOC 2013 Workshops*, volume 8377 of *Lecture Notes in Computer Science*, pp. 69–81. Springer International Publishing, 2014. doi:10.1007/978-3-319-06859-6_7. URL http://dx.doi.org/10.1007/978-3-319-06859-6_7. (Cited on page 13)

[ADEA10] D. Agrawal, S. Das, A. El Abbadi. Big Data and Cloud Computing: New Wine or Just New Bottles? *Proc. VLDB Endow.*, 3(1-2):1647–1648, 2010. doi:10.14778/1920841.1921063. URL http://dx.doi.org/10.14778/1920841.1921063. (Cited on page 13)

[AH00] E. Adar, B. A. Huberman. Free riding on Gnutella. *First Monday*, 5(10), 2000. (Cited on page 17)

[BA03] R. Blumberg, S. Atre. The problem with unstructured data. *DM REVIEW*, 13:42–49, 2003. (Cited on page 13)

[BAH+06] R. Blanco, N. Ahmed, D. Hadaller, L. A. Sung, H. Li, M. A. Soliman. A survey of data management in peer-to-peer systems. *University of Waterloo, Technical Report CS-2006-18*, 2006. (Cited on page 17)

[BKL08] R. Bryant, R. H. Katz, E. D. Lazowska. Big-Data Computing: Creating Revolutionary Breakthroughs in Commerce, Science and Society, 2008. (Cited on page 13)

[BL94] J. A. Boyan, M. L. Littman. Packet routing in dynamically changing networks: A reinforcement learning approach. *Advances in neural information processing systems*, pp. 671–671, 1994. (Cited on page 21)

[BQV05] R. Baldoni, L. Querzoni, A. Virgillito. Distributed Event Routing in Publish/Subscribe Communication Systems: a Survey. Technical report, 2005. (Cited on pages 20 and 21)

[CCK88] C. Cheng, I. Cimet, S. Kumar. A Protocol to Maintain a Minimum Spanning Tree in a Dynamic Topology. *SIGCOMM Comput. Commun. Rev.*, 18(4):330–337, 1988. doi:10.1145/52325.52357. URL http://doi.acm.org/10.1145/52325.52357. (Cited on page 25)

[CSC+06] A. Capocci, V. D. Servedio, F. Colaiori, L. S. Buriol, D. Donato, S. Leonardi, G. Caldarelli. Preferential attachment in the growth of social networks: The internet encyclopedia Wikipedia. *Physical Review E*, 74(3):036116, 2006. (Cited on page 13)

[CSWH01] I. Clarke, O. Sandberg, B. Wiley, T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing Privacy Enhancing Technologies*, pp. 46–66. Springer, 2001. (Cited on page 17)

[CYTZ09] X. Chen, W. Yin, P. Tu, H. Zhang. Weighted k-Means Algorithm Based Text Clustering. In *Proceedings of the 2009 International Symposium on Information Engineering and Electronic Commerce*, IEEC '09, pp. 51–55. IEEE Computer Society, Washington, DC, USA, 2009. doi:10.1109/IEEC.2009.17. URL http://dx.doi.org/10.1109/IEEC.2009.17. (Cited on page 51)

[DCCC05] J. Dowling, E. Curran, R. Cunningham, V. Cahill. Using feedback in collaborative reinforcement learning to adaptively optimize MANET routing. *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on*, 35(3):360–372, 2005. (Cited on page 56)

[GBMP13] J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami. Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013. (Cited on page 13)

[GR12] J. Gantz, D. Reinsel. The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east. *IDC iView: IDC Analyze the Future*, 2012. (Cited on page 13)

[Han03] U. Hansmann. *Pervasive Computing: The Mobile World*. Springer Professional Computing. Springer, 2003. URL http://books.google.de/books?id=rj6xT6BNakMC. (Cited on page 13)

[HE03] G. Hamerly, C. Elkan. Learning the K in K-Means. In *In Neural Information Processing Systems*, p. 2003. MIT Press, 2003. (Cited on pages 32 and 61)

[Her98] I. T. Hernádvölgyi. Generating random vectors from the multivariate normal distribution. Technical report, University of Ottawa, 1998. (Cited on page 46)

[HF10] T. Hu, Y. Fei. QELAR: A Machine-Learning-Based Adaptive Routing Protocol for Energy-Efficient and Lifetime-Extended Underwater Sensor Networks. *Mobile Computing, IEEE Transactions on*, 9(6):796–809, 2010. doi:10.1109/TMC.2010.28. (Cited on page 21)

[Jac09] A. Jacobs. The Pathologies of Big Data. *Commun. ACM*, 52(8):36–44, 2009. doi:10.1145/1536616.1536632. URL http://doi.acm.org/10.1145/1536616.1536632. (Cited on pages 13 and 14)

[KDT13] B. Koldehofe, F. Dürr, M. A. Tariq. Event-based systems meet software-defined networking. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS)*. 2013. (Cited on page 20)

[KV06] J. R. Kok, N. Vlassis. Collaborative Multiagent Reinforcement Learning by Payoff Propagation. *JOURNAL OF MACHINE LEARNING RESEARCH*, 7:1789–1828, 2006. (Cited on pages 21 and 22)

[LCP$^+$05] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, S. Lim. A Survey and Comparison of Peer-to-Peer Overlay Network Schemes. *IEEE Communications Surveys and Tutorials*, 7:72–93, 2005. (Cited on page 17)

[LKR04] J. Liang, R. Kumar, K. W. Ross. Understanding kazaa. *Manuscript, Polytechnic Univ*, 2004. (Cited on page 17)

[LLS04] M. Li, W.-C. Lee, A. Sivasubramaniam. Semantic small world: An overlay network for peer-to-peer search. In *Network Protocols, 2004. ICNP 2004. Proceedings of the 12th IEEE International Conference on*, pp. 228–238. IEEE, 2004. (Cited on page 18)

[Loh12] S. Lohr. The age of big data. *New York Times*, 11, 2012. (Cited on page 13)

[MJ09] A. Montresor, M. Jelasity. PeerSim: A Scalable P2P Simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pp. 99–100. Seattle, WA, 2009. (Cited on page 65)

[PHL04] L. Parsons, E. Haque, H. Liu. Subspace clustering for high dimensional data: a review. *ACM SIGKDD Explorations Newsletter*, 6(1):90–105, 2004. (Cited on page 62)

[RD01] A. I. T. Rowstron, P. Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the 18th IFIP/ACM international conference on distributed systems platforms*, pp. 329–350. Springer-Verlag, 2001. (Cited on page 18)

[RFH$^+$01] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A scalable content-addressable network. *ACM SIGCOMM Computer Communication Review*, 31:161–172, 2001. (Cited on page 18)

[SMLN$^+$03] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, H. Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for Internet applications. *IEEE/ACM Transactions on Networking*, 11:17–32, 2003. (Cited on page 18)

[TKK$^+$11] M. A. Tariq, B. Koldehofe, G. G. Koch, I. Khan, K. Rothermel. Meeting subscriber-defined QoS constraints in publish/subscribe systems. *Concurrency and Computation: Practice and Experience*, 23:2140–2153, 2011. (Cited on page 20)

[TKKR09] M. A. Tariq, B. Koldehofe, G. Koch, K. Rothermel. Providing Probabilistic Latency Bounds for Dynamic Publish/Subscribe Systems. In *Proceedings of the 16th ITG/GI conference on kommunikation in verteilten systemen (KiVS)*. Springer, 2009. (Cited on page 20)

[TKKR12] M. A. Tariq, B. Koldehofe, G. G. Koch, K. Rothermel. Distributed Spectral Cluster Management: A Method For Building Dynamic Publish/Subscribe Systems. In *Proceedings of the 6th ACM international conference on distributed event-based systems (DEBS)*. 2012. (Cited on page 20)

[TKR13] M. A. Tariq, B. Koldehofe, K. Rothermel. Efficient content-based routing with network topology inference. In *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems (DEBS)*. 2013. (Cited on page 20)

[TKR14] M. A. Tariq, B. Koldehofe, K. Rothermel. Securing Broker-Less Publish/Subscribe Systems Using Identity-Based Encryption. *IEEE Transactions on Parallel and Distributed Systems*, 25(2):518–528, 2014. doi:http://doi.ieeecomputersociety.org/10.1109/TPDS.2013.256. (Cited on page 20)

[VF05] M. Verleysen, D. François. The curse of dimensionality in data mining and time series prediction. In *Computational Intelligence and Bioinspired Systems*, pp. 758–770. Springer, 2005. (Cited on pages 15, 28 and 60)

[WHH96] M. H. Wl, M. E. Harmon, S. S. Harmon. Reinforcement Learning: A Tutorial, 1996. (Cited on page 21)

[WZWD14] X. Wu, X. Zhu, G.-Q. Wu, W. Ding. Data mining with big data. *Knowledge and Data Engineering, IEEE Transactions on*, 26(1):97–107, 2014. (Cited on page 13)

[YL03] L. Yu, H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. pp. 856–863. 2003. (Cited on page 62)

All links were last followed on July 12, 2014.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

place, date, signature