Fakultät Informatik, Elektrotechnik und Informationstechnik
Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Studienarbeit Nr. 2352

Elastische Last-Balancierung in einem Verteilten Semantischen Cache Overlay

Anja Reuter

Studiengang:	Informatik
Prüfer:	Prof. Dr. Bernhard Mitschang
Betreuer:	Dipl.-Inf. Carlos Lübbe
begonnen am:	24. Oktober 2011
beendet am:	24. April 2012
CR-Klassifikation:	H.2.4, H.2.8

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die immer stärkere Verbreitung von Smartphones mit high-speed Internetzugang führt dazu, dass Informationen immer und von überall abrufbar sind. Dies führt zu einer zunehmenden Popularität standortbezogener Dienste.

Standortbezogene Dienste berücksichtigen den aktuellen Standort des Nutzers und bieten Informationen, die im Kontext zur räumlichen Region stehen. Ein Beispiel wäre eine Umgebungssuche nach Restaurants, Tankstellen, oder Sehenswürdigkeiten, die über ein Smartphone abgeschickt wird.

Mit zunehmender Popularität eines standortbezogenen Dienstes erhöhen sich auch die Zugriffszahlen auf die Daten. Durch Großereignisse wie z.B. Konzerte, Sportveranstaltungen oder Demonstrationen, bei denen viele Menschen an einem Ort zusammentreffen, können zudem zeitlich und räumlich begrenzte Anfrage-Hotspots entstehen.

Ein Cache-Overlay für räumliche Daten, dass einen effizienten Zugriff auf die benötigten Daten bietet, kann die Last am Daten-Back-End reduzieren. Um die in Anfrage-Hotspots anfallende Last in einem Netz aus Cache-Knoten verteilen zu können, ist eine möglichst flexible Anpassung des Cache-Overlays an die aktuell anliegende Anfragelast nötig [LRM12].

In dieser Arbeit wurde die elastische Last-Balancierung in dem verteilten semantischen Cache Overlay DiSCO (Distributed Spatial Cache Overlay) aus [LRM12] weiterentwickelt. Zunächst wurde ein bestehender Mechanismus erweitert, der auf dem physikalischen Modell eines Feder-Partikel-Systems beruht.

Für die Verarbeitung von Anfragen wurden außerdem zwei neue Modi eingeführt, nach denen bestimmt wird, welche Knoten im Netzwerk an der

Beantwortung einer Anfrage beteiligt sind. So wird im Modus Niedrigste Last die Auslastung der einzelnen Knoten berücksichtigt und im kooperativen Modus erfolgt die Anfragebearbeitung unter Berücksichtigung des Cache-Inhalts benachbarter Knoten.

Für das Hinzufügen neuer Knoten zum Netzwerk sind in der Feder-Partikel-Topologie wie in [LRM12] beschrieben, besondere Maßnahmen zur Erhaltung der Stabilität des Netzwerks nötig. Daher wurde ein neuer Mechanismus zur Last-Balancierung entwickelt, der auf dem Konzept einer Schwerpunkt-Voronoi-Delaunay-Triangulierung beruht. In diesem Kontext wurden auch Mechanismen zum Hinzufügen neuer Knoten zum Netzwerk, sowie zur Aktualisierung der Topologie des Cache-Overlays entworfen.

Die Arbeit ist wie folgt strukturiert: In Kapitel 2 wird zunächst das zugrundeliegende System vorgestellt. Dann werden in Kapitel 3 die eingeführten Mechanismen erläutert und in Kapitel 4 wird die Implementierung der Mechanismen dargestellt. Das Kapitel 5 bietet eine Zusammenfassung der Arbeit, sowie einen Überblick über den Eigenanteil an den vorgestellten Komponenten.

2 Grundlagen des verteilten Cachings

Als Anwendungsgebiet für das in dieser Arbeit beschriebene Cache-Overlay, gelten standortbezogene Dienste, die auf ein Daten-Back-End mit geographischen Daten zugreifen.

Der Fokus im Bezug auf den Zugriff auf das Daten-Back-End liegt hier beim lesenden Zugriff auf die Daten. Der schreibende Zugriff auf das Daten-Back-End wird nicht näher betrachtet.

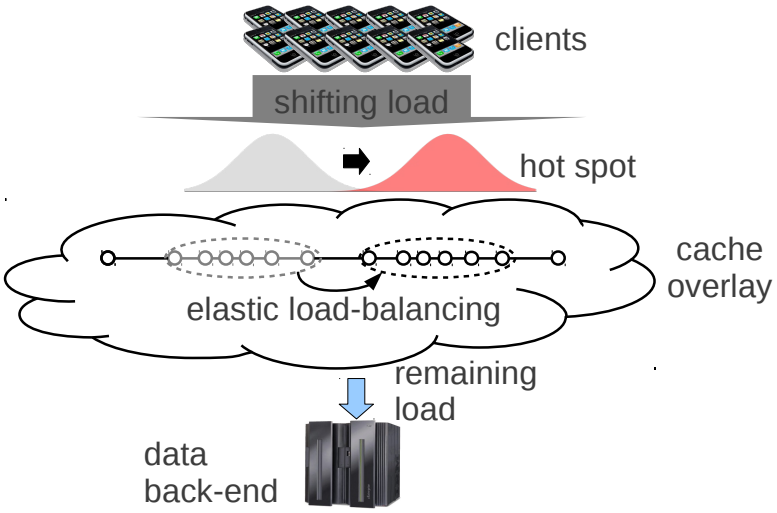
Es wird davon ausgegangen, dass die am Netzwerk beteiligten Rechner/Knoten permanent online sind und eine ausreichende Bandbreite vorhanden ist. Außerdem wird angenommen, dass die Übertragungslatenzen zwischen den Rechnern in etwa gleich groß sind.

Die Clients (z.B. Smartphones) greifen wie in Abb. 2.1 skizziert auf das Cache Overlay zu und das Overlay beantwortet die Anfragen und greift ggf. auf die Daten aus dem Daten-Back-End zu.

2.1 Verteiltes Caching räumlicher Daten

Das verwendete Schema für das Caching von räumlichen Daten basiert auf dem objekt-orientierten Datenmodell Nexus [NM04]. Ein Objekt wird dort als eine Menge an Attributinstanzen definiert. Jede Attributinstanz ist ein Tupel aus einem Attributnamen und -wert.

Wie in [LBC+11] beschrieben werden die Anfragen auf der Basis dieses Datenmodells, als Boolesche Ausdrücke über Prädikate formuliert. Ein Prädikat wird dabei als $A\phi C$ ausgedrückt, wobei A ein Attributname, C



Elastische Last-Balancierung zur Anpassung an Hotspots [LRM12]

Abbildung 2.1: Elastische Last-Balancierung

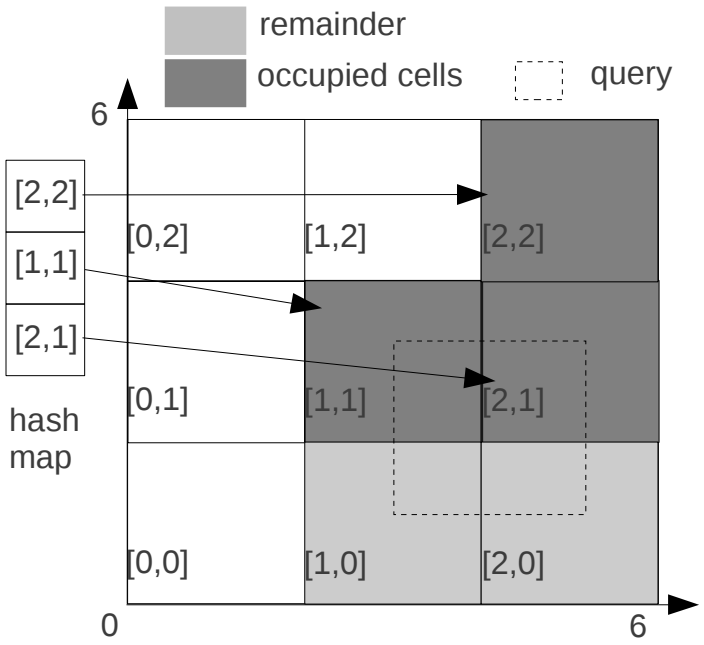
eine Konstante und ϕ ein Vergleichsoperator ist. Die Vergleiche können auch über räumliche Datenbereiche (*within*, *intersects*, *etc.*) formuliert sein.

Auf der Basis einer solchen Algebra wurde in [DFJ+96] ein Ansatz für einen semantischen Cache vorgestellt, in welchem der Cache-Inhalt logisch mit Hilfe eines Prädikates beschrieben wird.

Basierend auf dieser Beschreibung des Cache-Inhalts, kann jede gegebene Anfrage in die folgenden zwei disjunkten Teile aufgespalten werden:

- die *Probe Query* beschreibt den Teil der Anfrage, der vollständig aus den im Cache gespeicherten Objekten beantwortet werden kann.
- die *Remainder Query* ist das entsprechende Gegenstück und deckt den Teil der Query ab, für den das Daten-Back-End angesprochen werden muss.

Für die Verwendung im Cache Overlay wurde dieser Ansatz verwendet, indem für die Beschreibung des Cache-Inhalts die ortsbezogene Eigenschaft der Anfragen ausgenutzt wurde. Da mit standortbezogenen Daten gearbei-



Gitterbasierter Cache für räumliche Daten [LRM12]

Abbildung 2.2: Gitterbasierter Cache

tet wird, bietet sich diese Eigenschaft für die Beschreibung des Cache-Inhalts an.

Der gewählte Ansatz zur Beschreibung basiert auf einem einheitlichen Gitter. Jede Gitterzelle repräsentiert eine entsprechende räumliche Region und Objekte die diese räumliche Region schneiden werden im Cache gespeichert. Wenn ein Objekt die Grenze einer Zelle schneidet, kann es von mehreren Zellen gecached werden.

Die Gitterzellen werden über Schlüssel identifiziert, welche die von der Zelle abgedeckte räumliche Region wiederspiegeln. Von einem Cache-Knoten besetzte Zellen, werden, wie in Abb. 2.2 gezeigt, nach ihrem Schlüssel in einer Hash-Map gespeichert.

Für die Gitterzellen wird eine feste Länge verwendet, sodass für einen gegebenen Schlüssel leicht die entsprechende räumliche Region berechnet werden kann. So reicht die Gitterzelle [1,0] in Abb. 2.2 in der x-Dimension

von 2 bis 4 und in der y-Dimension von 0 bis 2. Entsprechend können für eine gegebene räumliche Region alle Schlüssel der Zellen berechnet werden, die diese Region schneiden.

Diese Eigenschaft wird auch für die Verarbeitung von Anfragen genutzt. Der in Abb.2.2 dargestellte Anfragebereich schneidet vier Gitterzellen. Daraus lassen sich entsprechend vier Schlüssel ableiten. Um die Anfrage zu bearbeiten, prüft der Cache zunächst für jeden der vier Schlüssel, ob ein entsprechender Eintrag in der Hash-Map vorhanden ist. Falls es einen Eintrag gibt, können die entsprechenden Objekte zum vorläufigen Ergebnis hinzugefügt werden. Gibt es keinen Eintrag, so wird die Zelle zur *Remainder Query* hinzugefügt.

Bezogen auf das Beispiel in Abb. 2.2 bedeutet dies, dass die Zellen [1,1] und [2,1] zum vorläufigen Ergebnis hinzugefügt werden. Die Remainder Query umfasst die beiden Zellen [1,0] und [2,0], sie wird somit definiert als: $R :=$ extent *intersects* $G_{[1,0]} \vee$ extent *intersects* $G_{[2,0]}$, wobei $G_{[x,y]}$ für die räumliche Region der Zelle [x,y] steht.

Die *Remainder Query* wird an das Daten-Back-End geschickt und die vom Daten-Back-End zurückgesandten Objekte werden zu dem vorläufigen Ergebnis hinzugefügt. Als letzter Schritt wird das vorläufige Ergebnis noch mit der ursprünglichen Anfrage gefiltert, da durch die Größe des Gitters evtl. mehr Objekte geladen wurden, als für die Beantwortung der Anfrage nötig sind.

Ein verteilter semantischer Cache basierend auf den beschriebenen Konzepten wird aus einem Netzwerk von N Knoten entworfen. Jeder Knoten enthält eine Menge an gecacheten Objekten und eine Beschreibung des Cache-Inhalts.

Die Knoten haben eine begrenzte Kapazität, dh. die maximale Anzahl an Objekten im Cache ist begrenzt. Daher muss festgelegt werden, in welcher Reihenfolge die im Cache enthaltenen Zellen ersetzt werden sollen, wenn der Cache seine Kapazität erreicht hat und neue Daten geladen werden müssen.

Als Ersetzungskriterium erhält jeder Knoten einen Cache-Fokus. Der Cache-Fokus ist ein Punkt im Datenraum, der die Position des Knotens bestimmt. Jeder Knoten ersetzt Zellen in seinem Cache nach ihrem Abstand zu seinem

Cache-Fokus. D.h. Zellen, die weit vom Cache-Fokus entfernt sind werden bevorzugt durch neue Datenobjekte ersetzt.

Sind die Knoten über den Datenraum verteilt, wird somit der Datenraum in Zonen aufgeteilt, die von den verschiedenen Knoten bevorzugt im Cache gehalten werden. Die Cache-Zonen der einzelnen Knoten sind nicht disjunkt, dh. ein Datenobjekt kann auch von mehreren Knoten gecached werden.

3 Techniken zur elastischen Last-Balancierung

3.1 Topologie basierend auf einem Feder-Partikel-System

Für den Entwurf des Netzwerks aus Cache-Knoten muss eine grundlegende Topologie für das Cache Overlay gewählt werden. Die implementierte Topologie in DiSCO basiert auf dem physikalischen Modell eines Feder-Partikel-Systems.

Im Folgenden wird zunächst das zugrunde liegende physikalische Modell vorgestellt. Dann wird die Umsetzung der physikalischen Vorlage in eine Struktur für die Topologie des semantischen Cache Overlays DiSCO dargelegt.

3.1.1 Vom physikalischen Modell zur Netzwerk-Topologie

Ein Feder-Partikel-System besteht aus Partikeln, die durch Federn verbunden sind. Die Partikel können entweder frei beweglich oder fixiert sein. D.h. ein fixierter Punkt behält seine Position bei, während ein unfixierter Punkt beweglich ist und von den mit ihm verbundenen Federn in eine andere Position geschoben oder gezogen werden kann. Durch die von den Federn ausgeübten Kräfte werden die unfixierten Partikel in eine Position gezwungen, die dem niedrigsten Energiezustand für das Gesamtsystem entspricht (s. Abb. 3.1).

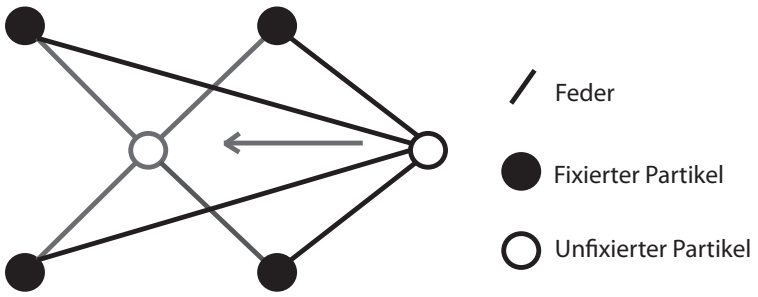


Abbildung 3.1: Federkräfte im Feder-Partikel-System

Wie in [LRM12] beschrieben, repräsentieren die Cache-Knoten im Cache Overlay die Partikel im Feder-Partikel-System und die Verbindungen zwischen benachbarten Knoten repräsentieren die Federn.

Wie in Kapitel 2 beschrieben, bestimmt der Cache-Fokus eines Knotens die Region, deren Daten bevorzugt von dem Knoten gecached werden. Außerdem bestimmt der Cache-Fokus die Position des Knotens in der Topologie des Cache Overlays.

In Hotspot-Regionen sollte die Dichte der Knoten möglichst hoch sein, um eine bessere Verteilung der Anfrage-Last zu ermöglichen (siehe auch Kapitel 3.2). Für eine optimale Verteilung der Knoten im Cache-Overlay müssen daher die Parameter im Feder-Partikel-Modell so gewählt werden, dass die Dichte der Knoten im Bereich von Hotspots hoch und in den anderen Bereichen des Netzwerks niedriger ist.

Im physikalischen Modell ist s die Streckung der Feder und die Federkonstante k gibt die Stärke der Feder wieder. Nach Hooke's Gesetz ist die Federkraft F proportional zur Stärke und zur Streckung, also $F = k \cdot s$.

Die Knoten im Cache Overlay sind durch Federn verbunden und die Streckung der Federn ist definiert als die Euklidische Distanz zwischen den Positionen (dem Cache-Fokus) der beiden verbundenen Knoten. In Regionen mit hoher Last werden die Federn zusammengedrückt um die Dichte der Knoten zu erhöhen. Entsprechend werden die Federn in Regionen mit niedriger Last entspannt.

Um diese Idee im Cache Overlay umzusetzen wurde die folgende Definition einer angepassten Federkonstante verwendet [LRM12]:

Die Definition der Gravitationskraft $G(n)$ eines Knotens ist so entworfen, dass die Arbeitslast des Knotens und die Dichte der Daten an der Position des Knotens berücksichtigt wird.

Die Dichte der Daten zu berücksichtigen ist wichtig, weil die Verteilung von räumlichen Daten in der Regel sehr unregelmäßig ist. Da die Knoten nur eine begrenzte Kapazität haben, kann ein Knoten mit einem Cache Fokus in einer Region mit hoher Datendichte nur eine sehr kleine Region des Datenraums abdecken. D.h. ein Knoten in Regionen mit geringer Datendichte kann viele Zellen in seinem Gitter ausfüllen und ein Knoten in einer Region mit hoher Datendichte nur wenige.

Daher wurde die Gravitationskraft eines Knotens definiert als:

$G(N) = 1 + (\rho(N) \cdot \alpha + load(N) \cdot (1 - \alpha)) \cdot \beta$

Wobei,

- $\rho(N)$ die Dichte der gespeicherten Daten ist (definiert als Anzahl der Objekte im Cache pro besetzter Zelle)
- $load(N)$ die Arbeitslast des Knotens als Anfragen pro Sekunde beschreibt
- $0 \leq \alpha \leq 1$ ein Parameter ist, mit dem die Gewichtung der beiden Aspekte der Last angepasst werden kann
- $0 \leq \beta \leq \infty$ ein Parameter ist mit dem der Einfluss der Gravitationskraft auf die Federkraft angepasst werden kann

Für eine Feder i zwischen einem Knoten N und seinen Nachbarn N_i, ist die Federkonstante definiert als die Summe der Gravitationskräfte der beiden Nachbarn: $k_i = G(N) + G(N_i)$.

Eine einzelne Feder i baut die Energie $E_i = \frac{1}{2} k_i s_i^2$ auf. Dies führt zur Gesamtenergie $\sum_i E_i = \sum_i \frac{1}{2} k_i s_i^2$ für das System. Das Minimum dieser Funktion bestimmt den niedrigsten Energiezustand für das Gesamtsystem.

Um den niedrigsten Energiezustand für das Gesamtsystem zu erreichen wird in [LRM12] ein einfacher Ansatz verwendet, der verteilt ausgeführt werden kann. Der Ansatz basiert auf der Idee, die Federn in jeder Iteration etwas zu entspannen, indem jeder Partikel einen kleinen Schritt in Richtung der Federkraft bewegt wird. Auf diese Weise werden die Federkräfte in jedem Schritt etwas reduziert und das System konvergiert gegen den stabilen Zustand der niedrigsten Gesamtenergie.

Der Einfluss der verschiedenen Parameter in der Definition der Gravitationskraft wurde in [LRM12] genauer untersucht.

3.1.2 Entwurf der Topologie für ein Elastisches Cache Overlay

Die verwendeten Cache-Knoten im Netzwerk sollen ihre Position an die Hotspots anpassen und sind daher im Feder-Partikel-Modell als unfixierte Partikel modelliert.

Da sich ein System, welches nur aus unfixierten Punkten und Federn besteht unweigerlich zu einem Punkt zusammenzieht, werden zusätzlich zu den Cache-Knoten noch Fixierungspunkte benötigt, welche das Netzwerk aufgespannt halten.

Zu diesem Zweck wurden an den Rändern des Netzwerkes fixierte Punkte eingeführt. In den Routing-Tabellen der benachbarten Knoten werden dies Punkte als spezielle Fixpunkte gelistet. Dies führt dazu, dass sich die Cache-Knoten auch am Rande des Netzwerks bewegen können, ohne zu stark nach innen gezogen zu werden.

Die in [LRM12] implementierte Topologie bestand zunächst nur aus einem einfachen Gitter (s. Abb. 3.2).

Der erste Schritt zur Stabilisierung, der in dieser Arbeit realisiert wurde, war die Einführung von Querverbindungen (s. Abb. 3.3). Durch die Querverbindungen wirken sich Oszilationseffekte im Netz weniger stark aus und das Netz wird stabiler.

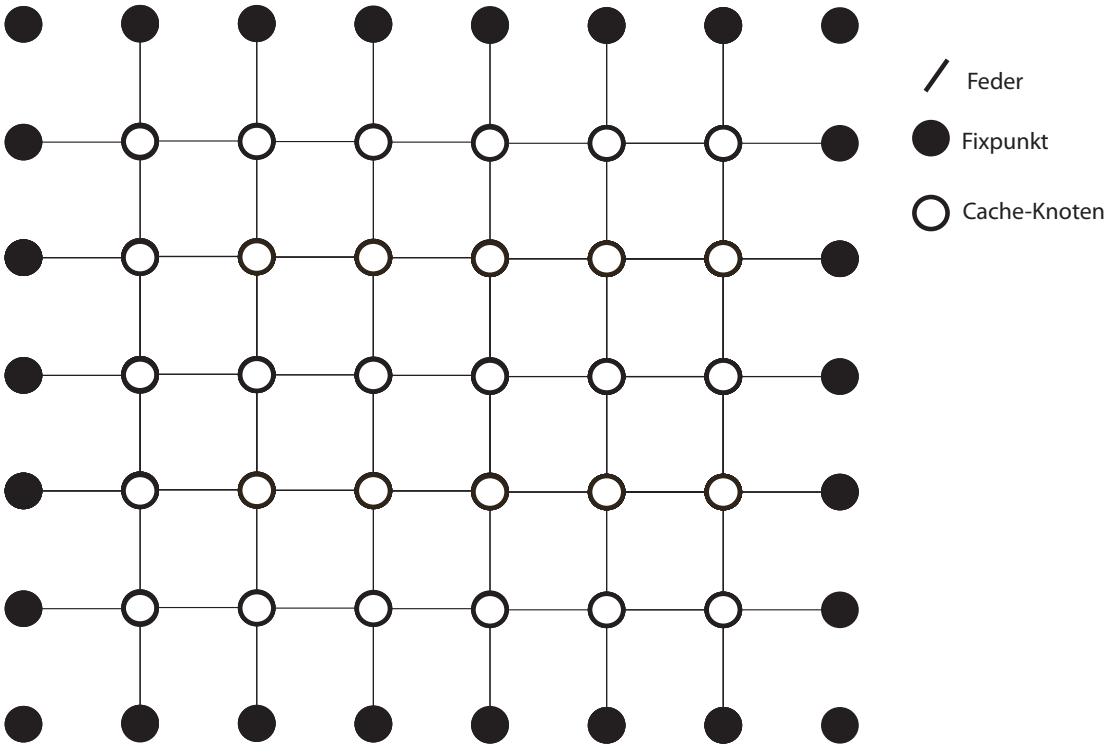


Abbildung 3.2: Topologie SimpleTopol

Beim Aufbau der Topologie werden zunächst die Knoten gleichmäßig in Reihen positioniert. Dabei wird über der obersten und unter der untersten Reihe von Knoten jeweils eine Reihe von Fixpunkten eingeführt. Zusätzlich wird am Anfang und am Ende jeder Reihe von Knoten ein Fixpunkt eingefügt. So entsteht ein Rechteck aus Knoten, welches an den Außenkanten von Fixpunkten umgeben ist.

Die Festlegung der Nachbarschaftsbeziehungen erfolgt in einem Durchlauf über die Knoten, sodass jeder Knoten mit den Knoten rechts und links von ihm, sowie über und unter ihm und auch mit den Knoten diagonal vom ihm verbunden wird.

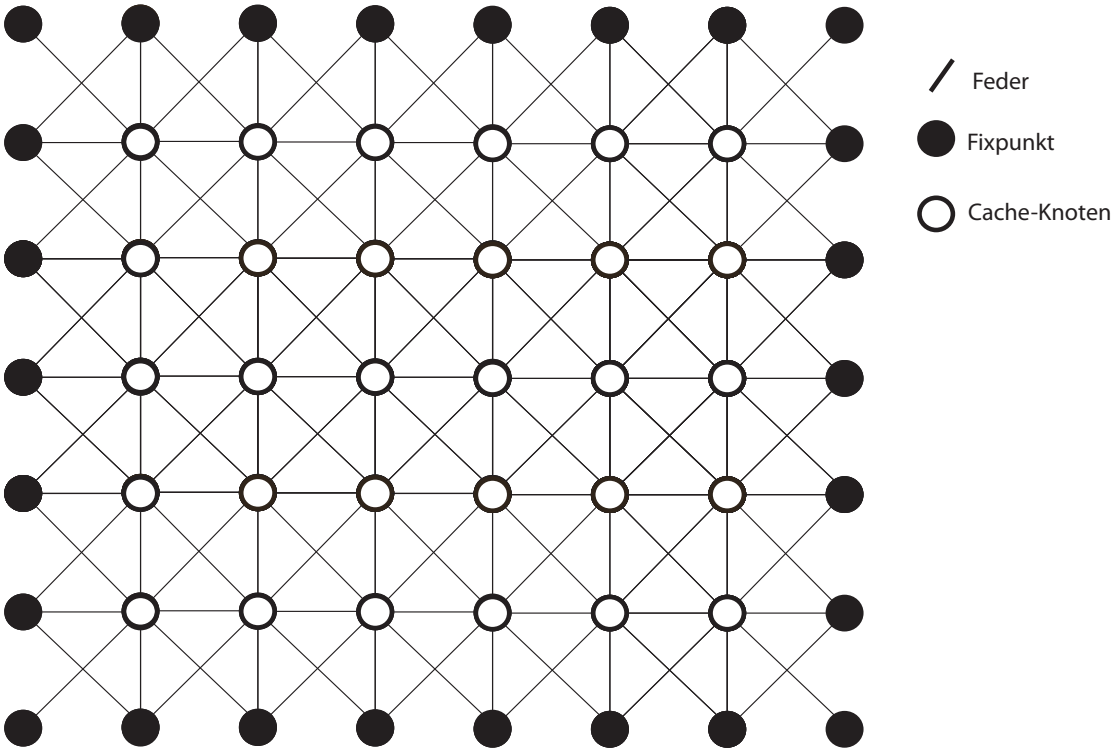


Abbildung 3.3: Topologie DiagonalTopol

3.2 Modi für die Anfrageverarbeitung

Als Bestandteil dieser Arbeit wurden für die Verarbeitung der Anfragen im Netzwerk zwei neue Verarbeitungsmodi eingeführt. Im Folgenden wird zunächst der in DiSCO implementierte Verarbeitungsmodus und dann die zwei neu eingeführten Modi erläutert.

Beim Eingang einer Anfrage in das Netzwerk, wird die Query zunächst an den Knoten weitergeleitet, dessen Cache-Fokus die geringste Entfernung zu der Region der Query hat. Dieser Knoten wird im Folgenden als Initiator-Knoten bezeichnet.

3.2.1 Initiator Modus

Im Initiator Modus werden die Anfragen von dem Knoten bearbeitet, dessen Cache-Fokus am nähesten an der Anfrageregion liegt (Initiator-Knoten). Da der Cache-Fokus die Region bestimmt, aus der bevorzugt Daten gecached werden, ist die Wahrscheinlichkeit bei diesem Knoten am höchsten, dass er die für die Anfrage relevanten Objekte bereits im Cache hat.

3.2.2 Modus Niedrigste Last

Im Modus Niedrigste Last verarbeitet der Initiator-Knoten die Anfrage selbst, solange eine vorgegebene maximale Last an dem Knoten nicht überschritten wird. Wird die Maximallast überschritten, so wird die Anfrage von dem Initiator-Knoten an denjenigen Nachbar-Knoten weitergereicht, welcher die geringste Last hat (s. Abb. 3.4).

Dies führt dazu, dass sich in Regionen mit hoher Last Cluster aus Knoten bilden. Die Last wird stärker zwischen den Knoten verteilt, was jedoch auch dazu führt, dass sich die Hitrate etwas verschlechtert, da der Knoten mit der geringsten Last nicht notwendigerweise die für die Anfrage relevanten Objekte im Cache hat.

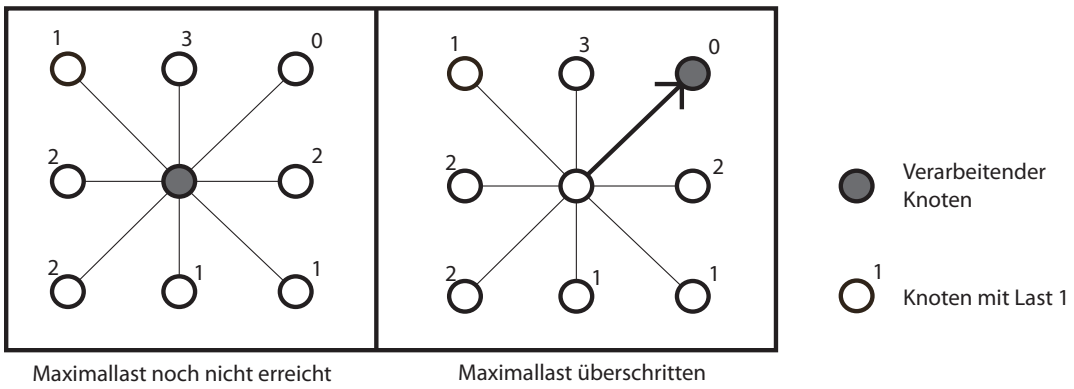


Abbildung 3.4: Schematische Darstellung des Modus Niedrigste Last

3.2.3 Kooperativer Modus

Die Grundidee beim kooperativen Modus besteht darin, dass die Nachbarknoten evtl. relevante Objekte für Anfragen gecached haben könnten, wenn die Anfrageregion groß ist und Teile der Anfrage näher am Cache-Fokus eines Nachbarn als an dem des Initiator-Knotens liegen.

Der Initiator-Knoten überprüft zunächst den eingenen Cache. Wenn der eigene Cache alle notwendigen Objekte zur Beantwortung der Anfrage enthält ist die Verarbeitung abgeschlossen.

Enthält der eigene Cache nicht alle notwendigen Objekte, so werden die Nachbarknoten für die Beantwortung der Anfrage hinzugezogen. Die Verarbeitung erfolgt nach folgenden, in Abb. 3.5 schematisch dargestellten Schritten:

Zunächst wird die Remainder Query nach folgendem Schema in Teilquerys aufgesplittet: Die Remainder Query wird repräsentiert als eine Liste an Schlüsseln zu Gitterzellen, die für die Anfragebeantwortung noch benötigt werden. Diese Liste wird aufgesplittet, indem für jeden Nachbarknoten N_i des Knotens eine Liste mit denjenigen Zellen angelegt wird, welche die geringste Entfernung zum Cache-Fokus von N_i haben. An jeden Nachbarn, dem auf diese Weise Zellen zugeordnet wurden, wird eine Teilanfrage mit den entsprechenden Zellschlüsseln geschickt.

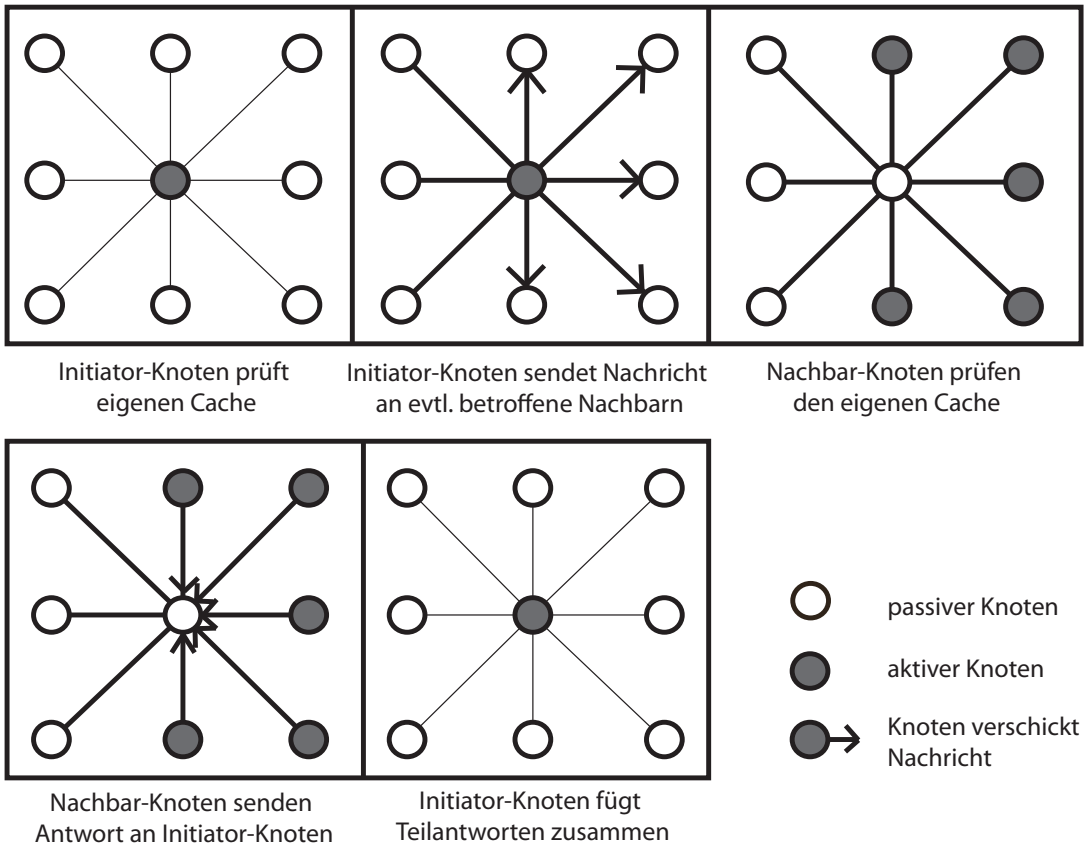


Abbildung 3.5: Schematische Darstellung des Modus Kooperativ

Bekommt ein Knoten eine Teilanfrage, so überprüft der Knoten den eigenen Cache auf die entsprechenden Zellen und schickt eine Anwort mit allen im Cache enthaltenen Zellen aus der Teilanfrage an den Initiator-Knoten. Der Initiator-Knoten sammelt die Antworten der Teilanfragen, bis von allen betroffenen Nachbarn eine Antwort eingetroffen ist.

Zum Schluss werden die Teilantworten zusammengesetzt. Falls noch Zellen für die Anfrageverarbeitung fehlen, wird eine Anfrage mit den fehlenden Zellen an das Daten-Back-End gestellt.

3.3 Topologie basierend auf einer Delaunay-Triangulierung

Die Topologie basierend auf einer Delaunay-Triangulierung wurde im Rahmen dieser Arbeit erstellt. Sie basiert auf dem Konzept eines Schwerpunkt-Voronoi-Diagramms und der entsprechenden Delaunay-Triangulierung.

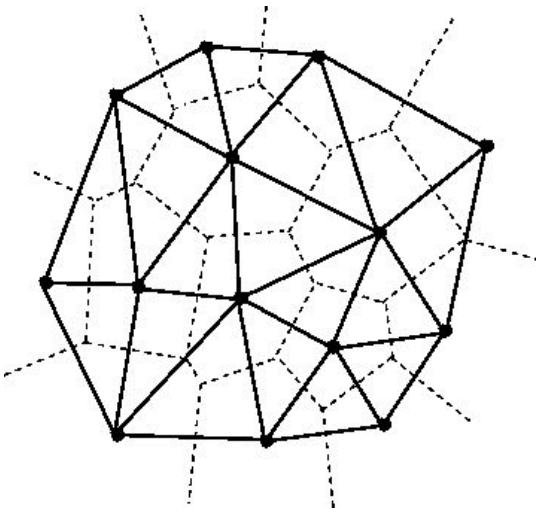
Voronoi Diagramme und Delaunay-Triangulierungen werden in einer Vielzahl von Anwendungen aus unterschiedlichen Forschungs- und Anwendungsfeldern verwendet. Eine umfassende Darstellung wird in [OBSC00] gegeben, einige Anwendungsbeispiele für Schwerpunkt-Voronoi-Diagramme werden in [DFG99] vorgestellt.

Im Folgenden wird zunächst das grundlegende Konzept eines Schwerpunkt-Voronoi-Diagramms erläutert. Dann wird die Übertragung des vorgestellten Konzepts auf die Netzwerk-Topologie dargestellt.

3.3.1 Schwerpunkt-Voronoi-Diagramm und Delauay-Triangulierung

In diesem Kapitel wird zunächst eine allgemeine Definition eines Voronoi-Diagramms und einer Delaunay-Triangulierung gegeben. Aufbauend darauf wird dann das Konzept eines dichtebasierten Schwerpunkt-Voronoidiagramms dargelegt.

Bezogen auf eine gegebene Menge von Punkten $\{z_i\}_{i=1}^{k}$ aus einer Domäne $\Omega \subset \mathbb{R}^N$, besteht die Voronoi Region V_i zu einem Punkt z_i aus allen Punkten in Ω, die näher an z_i sind als an allen anderen Punkten in der Menge. Die Menge der Voronoi Regionen $\{V_i\}_{i=1}^{k}$ bildet eine Partition der Domäne Ω und wird als Voronoi Diagramm von Ω bezeichnet. Die Punkte $\{z_i\}_{i=1}^{k}$ werden generierende Punkte oder Generatoren genannt. Die Delaunay-Triangulierung der Punktemenge (s. Abb. 3.6) wird gebildet, indem diejenigen generierenden Punkte miteinander verbunden werden, deren Voronoi-Regionen aneinander angrenzen [DG02, Kle89, Aur91, OBSC00].



Delaunay-Triangulierung über einem Voronoi-Diagramm
(Voronoi-Diagramm in gepunkteten Linien) [Lae]

Abbildung 3.6: Voronoi-Diagramm und Delanay Triangulierung

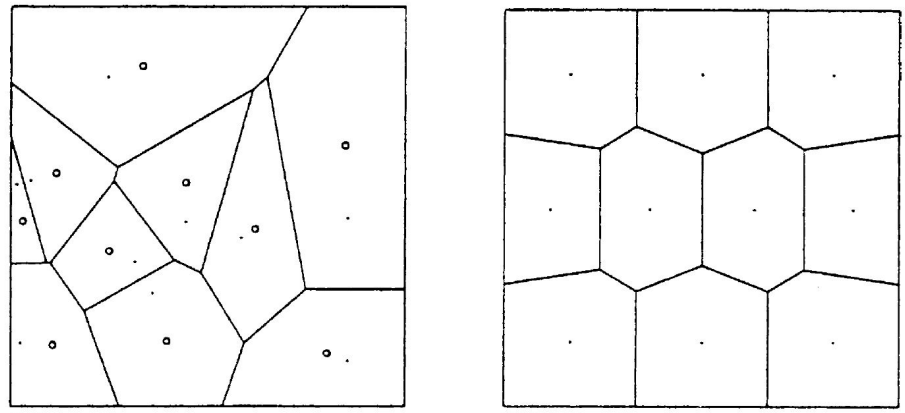
Ein Schwerpunkt-Voronoi-Diagramm ist ein Voronoi-Diagramm, in dem die generierenden Punkte jeweils im Schwerpunkt ihrer Voronoi Region liegen. Mit einer gegebenen Dichtefunktion $\rho(y)$ wird der Schwerpunkt $z*$ einer Region V definiert als:

$$z* = \frac{\int_V y\rho(y)dy}{\int_V \rho(y)dy} \tag{3.1}$$

Anhand einer gegebenen Punktemenge aus k Punkten $z*_i, i = 1, ..., k$ aus der Domänne Ω, können die entsprechenden Voronoi Regionen $V_i, i = 1, ..., k$ definiert werden und umgekehrt können anhand der Regionen $V_i, i = 1, ..., k$ die entsprechenden Schwerpunkte $z*_i, i = 1, ..., k$ bestimmt werden.

Gegeben eine Menge an Punkten $\{z_i\}_{i=1}^k$ in der Domäne Ω und eine positive Dichtefunktion ρ definiert in Ω, wird ein Voronoi Diagramm als Schwerpunkt-Voronoi-Diagramm bezeichnet, wenn

$$z_i = Z*_i \ , \ i = 1, ..., k \tag{3.2}$$



Ein Voronoi-Diagramm eines Quadrats mit 10 zufällig gewählten Punkten((a) (○) Schwerpunkte, (·) Generatoren) und einem 10-Punkt Schwerpunkt-Voronoi-Diagramm ((b) (·) Generatoren und Schwerpunkte) [DG02].

Abbildung 3.7: Voronoi-Diagramm und Schwerpunkt-Voronoi-Diagramm

d.h. die generierenden Punkte z_i sind die Schwerpunkte ihrer jeweiligen Voronoi Region. Die entsprechende Delaunay-Triangulierung wird als Schwerpunkt-Voronoi-Delaunay-Triangulierung bezeichnet [DG02].

Das diese Eigenschaft nicht generell auf Voronoi-Diagramme mit zufällig gewählten Punkten $\{z_i\}_{i=1}^{k}$ zutrifft zeigt die Abbildeng 3.7.

3.3.2 Algorithmus zur Berechnung einer Schwerpunkt-Voronoi-Delaunay-Triangulierung

Die Algorithmen zur Ermittlung von Schwerpunkt-Voronoi-Delaunay-Triangulierungen bestehen für gewöhnlich aus zwei Schritten:

1. Berechnung der generierenden Punkte für das Schwerpunkt-Voronoi-Diagramm
2. Konstruktion der entsprechenden Delaunay-Triangulierung

Für die Konstruktion der Delaunay-Triangulierung können bereits vorhandene Standardalgorithmen verwendet werden. Das Augenmerk richtet sich daher auf die Berechnung der generierenden Punkte für das Schwerpunkt-Voronoi-Diagramm.

Zur Berechnung der generierenden Punkte gibt es zum einen deterministische und zum anderen wahrscheinlichkeitsbasierte Verfahren. In dieser Arbeit wird zunächst eine wahrscheinlichkeitsbasierte K-Means Methode von MacQueen [Mac67] vorgestellt. Die Beschreibung richtet sich dabei nach [DG02], wo diese Methode zur Berechnung eines Schwerpunkt-Voronoi-Diagramms verwendet wird.

Gegeben ist eine Menge Ω, eine positive ganze Zahl k, und eine wahrscheinlichkeitsbasierte Dichtefunktion ρ definiert auf $\overline{\Omega}$. Die Berechnung der generierenden Punkte erfolgt nach dem folgenden Schema:

1. wähle eine Startmenge aus k Punkten $\{z_i\}_{i=1}^{k}$ in Ω und initialisiere den Index $j_i = 1$ für alle $i = 1, ..., k$
2. wähle ein zufälliges $y \in \Omega$ nach der wahrscheinlichkeitsbasierten Dichtefunktion $\rho(y)$
3. finde den Punkt z_n aus $\{z_i\}_{i=1}^{k}$ der am nähesten zu y liegt
4. setze den Punkt z_n auf

$$z_n = \frac{j_n z_n + y}{j_n + 1} \; und \; j_n = j_n + 1$$

5. wenn die so veränderte Punktemenge ein vorgegebenes Konvergenzkriterium erfüllt, erstelle die entsprechende Delaunay-Triangulierung und terminiere, ansonsten gehe zurück zum 2. Schritt.

Zentraler Gedanke des Algorithmus ist, dass die Punkte jeweils im Zentrum eines Clusters liegen. Das Cluster besteht aus Punkten, die nach der wahrscheinlichkeitsbasierten Dichtefunktion gewählt werden. D.h. für Regionen mit hoher Dichte werden viele Punkte erzeugt und für Regionen mit geringer Dichte werden entsprechend weniger Punkte erzeugt.

Die Cluster zu Beginn des Algorithmus bestehen aus einer Menge von k zufällig gewählten Punkten, die jeweils einen Clusterschwerpunkt repräsentieren.

Daraufhin werden nach und nach neue Punkte nach der wahrscheinlichkeitsbasierten Dichtefunktion erzeugt. Für jeden erzeugten Punkt wird zunächst das Cluster bestimmt zu dem er gehört. D.h. es wird der Punkt gewählt der dem erzeugten Punkt am nächsten liegt.

Dann wird der neue Punkt zu dem Cluster hinzugefügt und der Schwerpunkt des Clusters wird neu berechnet. Dabei wird der alte Schwerpunkt entsprechend der Anzahl der bereits erfolgten Aktualisierungen j_i gewichtet.

Dieses Vorgehen wird so lange wiederholt, bis ein vorgegebenes Konvergenzkriterium erfüllt ist.

3.3.3 Aufbau der Topologie des Cache Overlays

Die Grundidee ist, dass das Netzwerk als Voronoi-/Delaunay-Diagramm aufgebaut wird. Die Knoten sollen dabei die generierenden Punkte in einem Voronoi-Diagramm sein.

Die Anfragebearbeitung wird in dem Knoten durchgeführt, dessen Cache-Fokus die gerinste Entfernung zur Anfrageregion hat. D.h. der Knoten erhält Anfragen, die in seiner Voronoi-Region liegen. Außerdem ersetzt der Knoten vorrangig diejenigen Zellen in seinem Cache, welche weit von seinem Cache-Fokus entfernt sind. Für ein Schwerpunkt-Voronoi-Diagramm bedeutet dies, dass Zellen mit größerem Abstand zum Schwerpunkt der Voronoi-Region vorrangig ersetzt werden.

Als Nachbarn eines Knotens N gelten die Knoten, deren Voronoi-Region an die Region von N angrenzt. D.h. die Nachbarschaftsbeziehungen im Netzwerk entsprechen der Delaunay-Triangulierung über die Knoten.

Zur Initialisierung der Netzwerktopologie werden zunächst alle Knoten des Netzwerks positioniert. Als Anfangsposition wurde hier zunächst eine

gleichmäßige Verteilung gewählt, die der Verteilung aus der in Kapitel 3.1 beschriebenen Topologie entspricht.

Dann wird über die Knoten des gesamten Netzwerks eine Delaunay-Triangulierung erstellt. Dafür wurde das Computational Geometry Paket aus [Baro8] verwedet. Nach dieser Triangulierung werden die Nachbarschaftsbeziehungen unter den Knoten initialisiert. Auf diese Weise erhält jeder Knoten im Netzwerk eine Routing-Tabelle mit den benachbarten Knoten.

Da die weitere Verarbeitung im Netzwerk verteilt erfolgen soll, wird in jedem Knoten eine eigene Triangulierung auf der Basis der in der Routing-Tabelle enthaltenen Nachbarknoten erstellt. Die Vereinigung der Triangulierungen in den einzelnen Konten entspricht der ursprünglichen Triangulierung über das gesamte Netzwerk, da jeder Knoten seine Nachbarn zunächst aus dieser Triangulierung erhalten hat.

3.3.4 Hinzufügen neuer Knoten

Beim Hinzufügen eines neuen Knotens zum Netzwerk, muss die Topologie entsprechend angepasst werden. D.h. die benachbarten Knoten müssen den neuen Knoten zu ihrer Triangulierung hinzufügen, ihre Routing-Tabelle entsprechend anpassen und ggf. die Verbindung zu nicht mehr direkt benachbarten Knoten entfernen.

Die Positionierung eines neuen Knotenpunktes erfolgt nach zufällig ausgewählten Koordinaten. Für das Hinzufügen dieses neuen Knotenpunktes werden zunächst die folgenden, in Abb. 3.8 skizzierten Schritte, durchlaufen:

- Eine Nachricht mit der Position des neuen Knotens N_{neu} wird an den Knoten $N_{nearest\_neighbor}$ im Netzwerk weitergeleitet, welche am nähesten an der Position des neuen Knotens liegt.
- Der Knoten $N_{nearest\_neighbor}$ leitet die Nachricht mit der Position von N_{neu} an seine Nachbarn weiter.

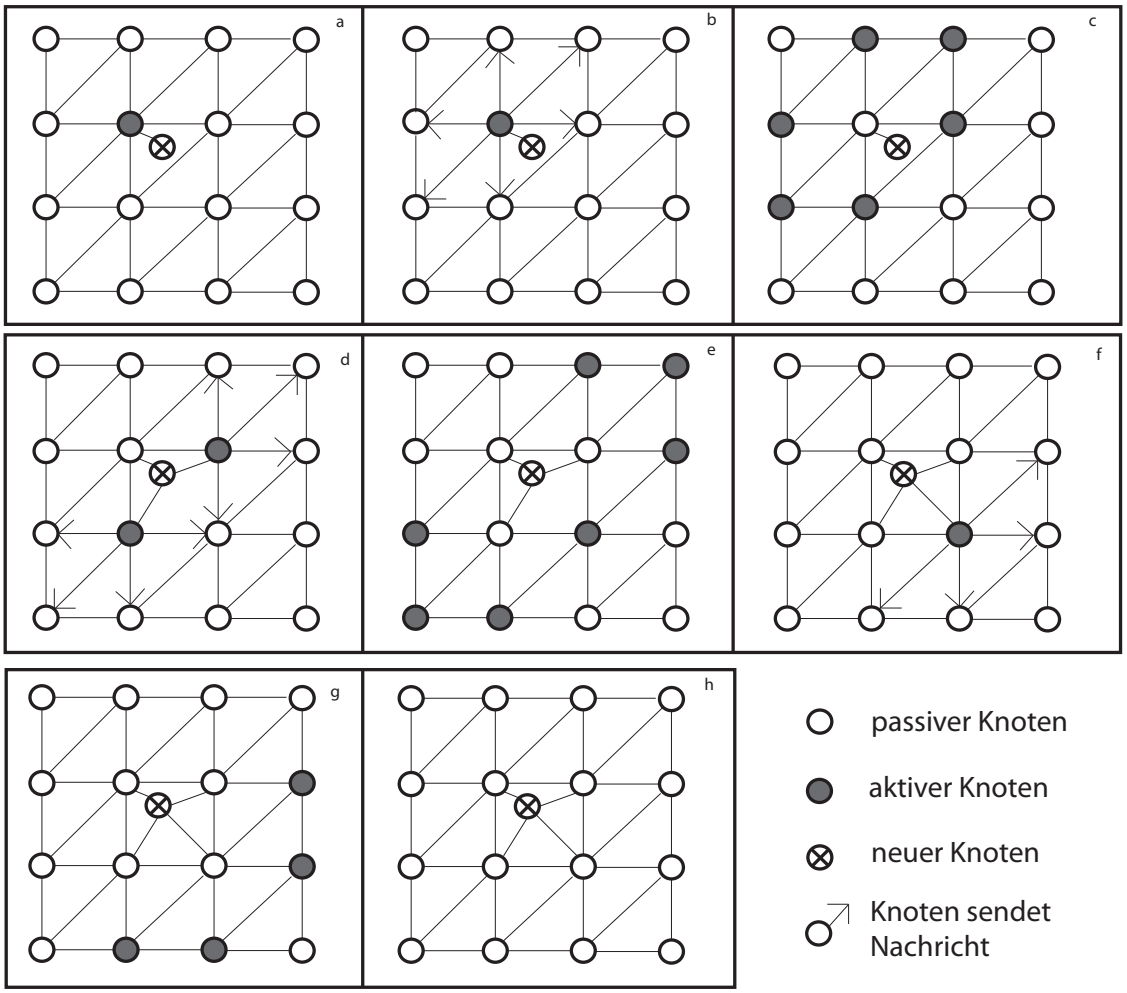


Abbildung 3.8: Hinzufügen eines neuen Knotens zum Netzwerk

- Der Knoten $N_{nearest\_neighbor}$ wird zu der Routing-Tabelle des Knotens N_{neu}hinzugefügt. (Da $N_{nearest\_neighbor}$ von allen Knoten im Netzwerk an nähesten an dem neuen Knoten liegt, muss es in der Triangulierung eine Verbindung zwischen den beiden Knoten geben.)
- Der Knoten $N_{nearest\_neighbor}$ fügt den neuen Knoten zu seiner Triangulierung hinzu, aktualisiert die Triangulierung und löst die Verbindung zu allen Knoten, die nach der neuen Triangulierung keine direkten Nachbarn von $N_{nearest\_neighbor}$ sind.

In jedem weiteren Knoten N_i, an welche die Nachricht weitergeleitet wird, erfolgt die Verarbeitung der Nachricht nach folgendem Vorgehen:

- Zuerst wird die Liste der aktuellen Nachbarn von N_i gespeichert.
- Dann wird der Knoten N_{neu} zur Triangulierung von N_i hinzugefügt und die Nachbarschaftsliste von N_i aktualisiert.
- Zuletzt werden die Nachbarschaftlisten vor und nach dem Update verglichen und
  - zu den Nachbarn, die in der alten, aber nicht in der neuen Liste enthalten sind, wird die Verbindung gelöscht; außerdem wird die Nachricht an sie weitergeleitet (damit sie die Verbindung ebenfalls löschen)
  - zu den Nachbarn, die in der neuen, aber nicht in der alten Liste enthalten sind, wird eine Verbindung aufgebaut; außerdem leitet der Knoten die Nachricht noch an seine Nachbarn weiter, falls eine neue Verbindung aufgebaut wurde, da sich in diesem Fall die Triangulierung der Nachbarknoten ebenfalls ändern könnte.

Die Topologie ist somit genau dann vollkommen aktualisiert, wenn die Nachricht von keinem Knoten mehr weitergeleitet wird.

3.3.5 Umsetzung der Lastadaptivität

Der in Kapitel 3.3.2 beschriebene Algorithmus basiert auf einer Dichtefunktion, die für den den gesamten Datenraum anwendbar ist. Durch die verteilte Struktur des Cache Overlays lässt sich jedoch keine Dichtefunktion für das gesamte Netzwerk angeben, sondern immer nur für jeweils eine Node unter Berücksichtigung der Nachbarn.

Aus diesem Grund wurde der Algorithmus so angepasst, dass die Zufallspunkte nicht für das gesamte Netzwerk zentral erzeugt werden, sondern dass jeder Knoten eigene Zufallspunkte erzeugt. D.h. die Clusterbildung erfolgt nicht zentral, sondern jeder Knoten formt sein eigenes Cluster.

Eine Dichtefunktion für einen Knoten N unter Berücksichtigung der Last an den Nachbarknoten N_i kann z.B. wie folgt definiert werden:

$$\rho(x,y) = \frac{Last(N)}{d(N,(x,y))} + \sum \frac{Last(N_i)}{d(N_i,(x,y))} \tag{3.3}$$

Dies entspricht einer Mittelung der Last an den Knoten, gewichtet nach der Entfernung der Knoten zum betrachteten Punkt.

Eine weitere Schwierigkeit besteht in der Festlegung eines Konvergenzkriteriums. Die Dichtefunktion eines Knotens im Cache-Overlay richtet sich nach der aktuellen Last an dem Knoten und seinen Nachbarn. Daher ändert sich mit einer Laständerung auch die Berechnung der Dichte und somit die Lage der Schwerpunkte für das Voronoi-Diagramm.

Die Verschiebung der Schwerpunkte bei einer Laständerung ist genau der im Cache-Overlay erwünschte Effekt. Eine Konzentration von Schwerpunkten in Regionen mit hoher Last entspricht einer Erhöhnung der Knotendichte in Regionen mit hoher Last, da die Knoten des Netzwerks in den Schwerpunkten liegen.

Die Konsequenz für den verwendeten Algorithmus besteht darin, dass die Berechnung des Schwerpunkt-Voronoi-Diagramms nicht nach einem Konvergenzkriterium abgeschlossen werden kann, sondern durchgängig weitergeführt werden muss. Die Iteration über die Ermittlung des Clusterschwerpunktes wird daher permanent fortgesetzt.

Die Dichtefunktion bleibt bei konstanter Lastverteilung ebenfalls konstant. D.h. die Clusterbildung anhand der dichtebasierten Zufallspunkte führt in diesem Fall nur zu kleinen Verschiebungen des Clusterschwerpunktes.

Für die Anpassung der Knotenposition wird eine Mindestdistanz zwischen altem und neuem Clusterschwerpunkt festgelegt, sodass die Position des Knotens in diesem Fall nicht verändert wird. D.h. die Berechnung bleibt bei einer zu kleinen Veränderung des Clusterschwerpunkts ohne Auswirkung für die Knotenposition im Cache-Overlay.

Der angepasste K-Means Algorithmus für die Verwendung im Cache-Overlay verläuft für jeden Knoten N_i mit der Position p_i und der Dichtefunkton ρ_i im Overlay verteilt nach folgendem Schema:

1. wähle ein zufälliges $y \in \Omega$ nach der wahrscheinlichkeitsbasierten Dichtefunktion $\rho_i(y)$
2. berechne den Punkt p_{neu} als

$$p_{neu} = \frac{j_i p_i + y}{j_i + 1}$$

3. wenn die Distanz zwischen p_i und p_{neu} größer der Mindestdistanz ist, sezte $p_i = p_{neu}$ *und* $j_i = j_i + 1$ und markiere den Knoten als bewegt
4. wenn $j_i < max_j$, dann setze $j_i = j_i + 1$, sonst setze $j_i = 1$
5. gehe zurück zu Schritt 1.

Auch im angepassten Algorithmus wird der alte Schwerpunkt entsprechend der Anzahl der bereits erfolgten Aktualisierungen j_i gewichtet. Da die Interation jedoch unendlich fortgesetzt wird, würde j_i auch unendlich groß werden. Bei der Division durch $j_i + 1$ würde im Verlauf somit unweigerlich ein Unterlauf entstehen.

Um dies zu vermeiden wird ein Maximalwert für j_i eingeführt und wie im Schritt 4 beschrieben, wird $j_i = 1$ gesetzt wenn der Maximalwert erreicht wird. Bezogen auf die Clusterbildung bedeutet dies, dass ab dem Punkt, an dem es die maximale Anzahl an Verschmelzungen gegeben hat, die Clusterbildung von vorn beginnt.

Die unter Schritt 3 erwähnte Markierung des Knotens als bewegter Knoten dient als Anhaltspunkt für die Aktualisierung der Nachbarschaftsbeziehungen (s. Kapitel 3.3.6). Verändert ein Knoten seine Position, so besteht damit die Möglichkeit, dass die Verbindungen zu seinen Nachbarknoten danach keine Delaunay-Triangulierung der Knoten mehr darstellt und die Topologie aktualisiert werden muss.

3.3.6 Aktualisierung der Topologie des Overlays

Zur Aktualisierung der Nachbarschaftsbeziehungen zwischen den Knoten des Netzwerks wird in regelmäßigem Abstand eine Aktualisierungsnachricht an einen zufällig ausgewählten Knoten gesendet.

Erhält ein Knoten N eine Aktualisierungsnachricht, so prüft er zunächst, ob er als bewegt markiert wurde. Ist der Knoten nicht als bewegt markiert, so wird die Aktualisierungsnachricht von ihm ignoriert.

Wenn der Knoten N bewegt wurde, hat sich die Position des Knotens geändert und es besteht die Möglichkeit, dass die Nachbarschaftsbeziehungen des Knotens aktualisiert werden müssen, um wieder eine Delaunay-Triangulierung zu erhalten.

Wie in Kapitel 3.3.3 beschrieben, enthält jeder Knoten eine eigene Triangulierung über sich selbst und seine Nachbarknoten. Um festzustellen, ob eine Aktualisierung nötig ist, führt der Knoten N zunächst eine Aktualisierung der eigenen Triangulierung aus.

Dann wird überprüft, ob in der aktualisierten Triangulierung Knoten enthalten sind, die keine direkten Nachbarn von N sind. Falls dies der Fall ist, so werden die Nachbarschaftsbeziehungen des Knotens aktualisiert.

Die Aktualsierung der Nachbarschaftsbeziehungen des Knotens erfolgt, indem zunächst die Verbindung zu allen Nachbarn gelöst wird, mit Ausnahme des Nachbarn, der die geringste Entfernung zum Knoten N hat.

Dann wird an den verbliebenen Nachbarn eine Nachricht geschickt, die der Nachricht entspricht, die beim Hinzufügen eines neuen Knotens zum Netzwerk verschickt wird. Das weitere Vorgehen zur Erstellung der Verbindungen zwischen Nachbarknoten entspricht dem in Kapitel 3.3.4 beschiebenen Vorgehen beim Hinzufügen eines neuen Knotens zum Netzwerk.

4 Implementierung

In diesem Kapitel werden einige Bestandteile der Implementierung vorgestellt. Für die Simulation wurde der Peer-to-Peer Simulator PeerSim [JMJV10] verwendet. In diesem Simulator können Peer-to-Peer Overlays aus Knoten mit beliebigen Protokollen simuliert werden. Für die Simulation des Cache Overlays wurde PeerSim entsprechend erweitert [LRM12].

Die Darstellung in diesem Kapitel beschränkt sich auf die Bestandteile der Implementierung, die im Rahmen dieser Arbeit entstanden sind, oder an denen Erweiterungen oder Anpassungen vorgenommen wurden. Zum Verständnis der Funktionsweise werden zudem noch einige zentrale Komponenten beschrieben, die bereits vollständig implementiert waren.

Zu Beginn der Simulation wird zunächst die Topologie des Netzwerks aufgebaut. Im Verlauf der Simulation erfolgt die Verarbeitung von Anfragen und die Kommunikation innerhalb des Netzwerks über Protokolle in den Knoten und über Nachrichten, die zwischen den Knoten verschickt werden.

Durch eine entsprechende Konfigurationsdatei wird festgelegt, welche Klasse zum Aufbau des Netzwerks genutzt wird und welche Protokolle und Nachrichten im Netzwerk verwendet werden.

Im Folgenden werden zunächst die Klassen zum Aufbau der Topologie beschrieben. Danach werden die verwendeten Protokolle und anschließend die Nachrichten für die verschiedenen Topologien erläutert.

4.1 Aufbau der Netzwerk-Topologie

Im Folgenden werden die beiden Klassen zur Initialisierung der Topologie basierend auf einem Feder-Partikel-System (s. Kapitel 3.1) und der Topologie basierend auf einer Schwerpunkt-Voronoi-Delaunay-Triangulierung (s. Kapitel 3.3) beschriebenen.

4.1.1 Die Klasse SimpleTopol

Die Klasse SimpleTopol dient zur Initialisierung der Topologie basierend auf einem Feder-Partikel-System, welche in Kapitel 3.1 beschrieben wurde. Zur Auswahl stehen dabei die Topologien SIMPLE und DIAGONAL. Der verwendete Modus für die Topologie kann in der Konfiguationsdatei festgelegt werden. Der Modus SIMPLE war bereits implementiert und die Klasse wurde im Rahmen dieser Arbeit um den Modus DIAGONAL erweitert.

Zur Initialisierung der Topologie Diagonal wird zunächst eine Liste aus Knotenpunkten erzeugt. Ein Knotenpunkt kann dabei entweder einen Netzwerk-Knoten enthalten, oder als Knotenpunkt ohne Cache-Knoten erzeugt werden.

Knotenpunkte ohne Cache-Knoten sind die in Kapitel 3.1.2 bereits erwähnten Fixpunkte. Sie werden in den Routing-Tabellen der benachbarten Knotenpunkte geführt um die Fixpunkte für das Feder-Partikel-Modell zu simulieren. Da diese Knotenpunkte jedoch keine Cache-Knoten enthalten, spielen sie für die Verarbeitung von Anfragen und für das Versenden von Nachrichten im Netzwerk keine Rolle.

Die Knotenpunkte werden zeilenweise nach der gewünschten Overlay-Struktur positioniert und in die Liste eingefügt. D.h. für ein Netzwerk mit x Knoten in horizontaler und y Knoten in vertikaler Richtung wird zunächst eine Reihe aus $x + 2$ Fixpunkten eingefügt. Dann wird y-mal jeweils ein Fixpunkt, x Knotenpunkte mit Cache-Knoten und noch ein Fixpunkt hinzugefügt. Zum Abschluss folgt noch eine Reihe aus $x + 2$ Fixpunkten (s. Abb.4.1)

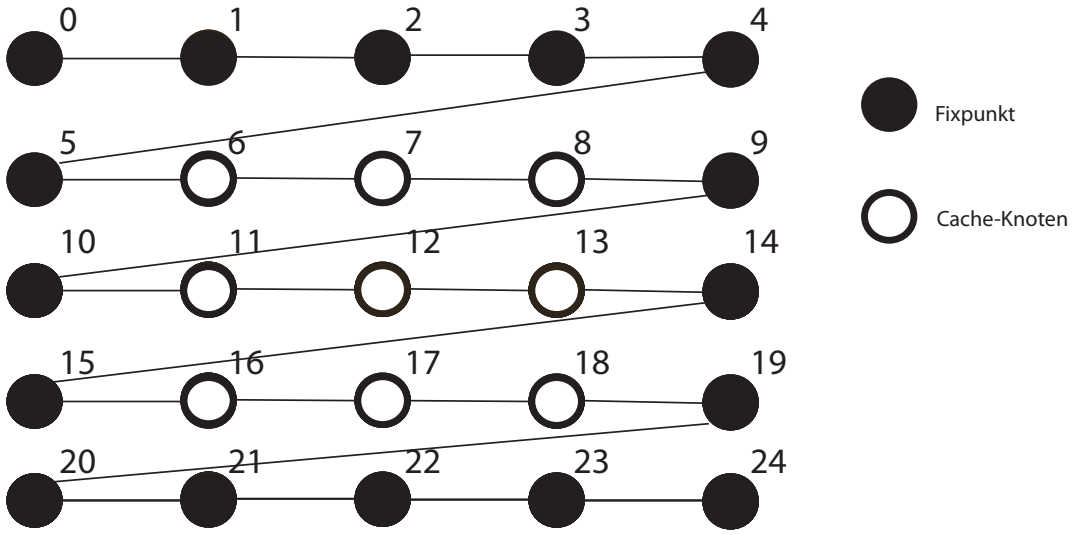


Abbildung 4.1: Liste der positionierten Knoten

Die Initialisierung der Nachbarschaftsbeziehungen zwischen den Knoten erfolgt in einem Durchlauf über die Liste der Knotenpunkte. Für jeden Knotenpunkt mit einem Cache-Knoten werden die benachbarten Knoten nach dem in Abb. 4.2 dargestellten Schema in die Routing Tabelle im Protokoll LAOLink (s. Kapitel 4.2.2) übernommen.

4.1.2 Die Klasse VoroTopol

Die Klasse VoroTopol wurde im Rahmen dieser Arbeit erstellt. Sie dient zur Initialisierung der Topologie basierend auf einer Schwerpunkt-Voronoi-Delaunay-Triangulierung, welche in Kapitel 3.3 beschrieben wurde.

Ähnlich wie in der Klasse SimpleTopol, wird zunächst eine Liste aus Knotenpunkten erstellt. Da für die Voronoi-Delaunay-Topologie jedoch keine Fixpunkte nötig sind, werden nur Knotenpunkte erzeugt, die auch einen Cache-Knoten enthalten.

Für die Positionierung der Knotenpunkte wurde zunächst das gleiche Schema verwendet, nach dem die Knoten in der Feder-Partikel-Topologie

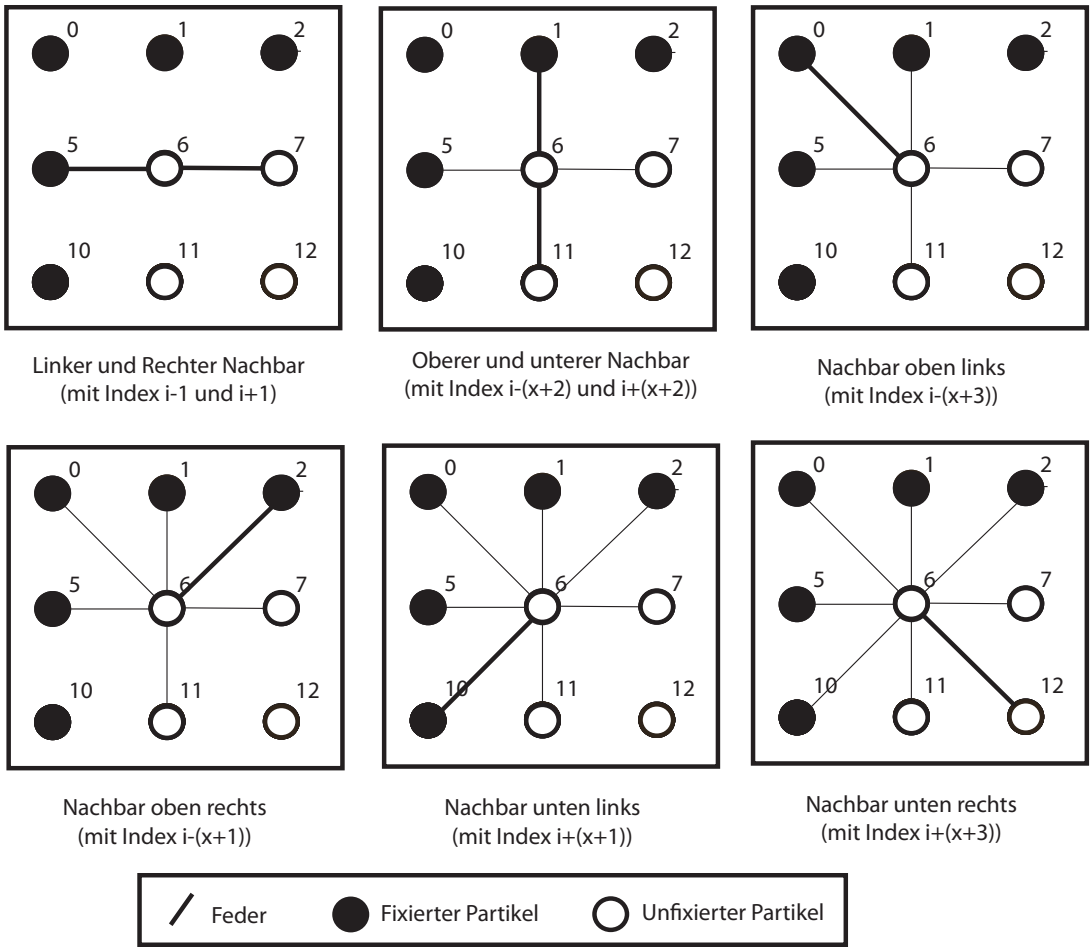


Abbildung 4.2: Festlegung der Nachbarschaftsbeziehungen

angeordnet wurden. Prinzipiell können jedoch auch beliebige andere Startpositionen implementiert werden.

Nachdem die Knotenpunkte positioniert wurden, wird eine Delaunay-Triangulierung über die Knotenpositionen erstellt. Für die Triangulierung wird das Computational Geometry Paket aus [Baro8] verwendet. Aus der Triangulierung wird eine Liste der Segmente der Triangulierung entnommen. Ein Segment besteht jeweils aus zwei Punkten, die in der Triangulierung verbunden sind.

Anhand der Liste der Segmente werden die Nachbarschaftsbeziehungen für das Netzwerk initialisiert, indem für jedes Segment beide Knoten den jeweils anderen Knoten in ihre Routing-Tabelle im Protokoll VoroLink (s. Kapitel 4.2.5) aufnehmen.

4.2 Protokolle

Die in diesem Kapitel erläuterten Protokolle liegen jeweils in den Knoten des Netzwerks und dienen zur Verarbeitung der eingehenden Nachrichten und zur Speicherung und Verwaltung von Informationen über den jeweiligen Knoten. Die verschiedenen Protokolle können über den übergeordneten Knoten und eine entsprechende Protokoll-ID angesprochen werden.

4.2.1 LAOProtocol

In dem Protokoll LAOProtocol werden die eingehenden Nachrichten eines Netzwerk-Knotens verarbeitet. Es werden Anfragen bearbeitet, HeartBeat-Nachrichten an die Nachbarn verschickt und die Position des Knotens angepasst.

Die im Rahmen dieser Arbeit vorgenommenen Änderungen am Protokoll LAOProtocol beziehen sich auf die im Kapitel 3.2 beschriebenen neu eingeführten Modi für die Verarbeitung von Anfragen.

Die eingehenden Anfragen werden in einer Route-Nachricht durch das Netzwerk zu dem Knoten transportiert, dessen Cache-Fokus am nähesten an der Anfrageregion liegt. D.h. eine Route-Nachricht wird im LAO-Protocol verarbeitet, indem geprüft wird, ob einer der Nachbarknoten einen Cache-Fokus hat, der näher an der Anfrageregion liegt als der Cache-Fokus des eigenen Knotens. Ist dies der Fall, so wird die Route-Nachricht weitergeleitet. Gibt es keinen Nachbarknoten, der näher an der Anfrageregion liegt, so wird die Anfrage in dem in der Konfigurationsdatei vorgegebenen Modus wie folgt verarbeitet:

Initiator Modus: Im Initiator Modus verarbeitet der Knoten die Anfrage selbst, d.h. die Anfrage wird aus der Route-Nachricht extrahiert und im CacheProtocol (s. Kapitel 4.2.3) verarbeitet.

Modus Niedrigste Last: Im Modus Niedrigste Last wird zunächst über das CacheProtocol geprüft, ob die aktuelle Last an dem Knoten über einem in der Konfigurationsdatei festgelegten Maximalwert liegt. Ist der Maximalwert noch nicht überschritten, so wird die Anfrage wie im Initiator Modus vom Knoten selbst bearbeitet.

Wenn die Last am Knoten den Maximalwert überschritten hat, erfolgt ein Durchlauf über die Nachbarknoten. Dabei wird der Nachbarknoten ermittelt, an welchem die geringste Last anliegt. An diesen Knoten wird dann die Anfrage zur Verarbeitung weitergeleitet.

Kooperativer Modus: Im kooperativen Modus wird die Anfrage von dem Knoten zunächst im CacheProtocol (s. Kapitel 4.2.3) so weit verarbeitet, dass ein vorläufiges Ergebnis gespeichert wird, welches folgende Komponenten enthält:

- Eine Liste mit den Objekten aus dem Cache, auf welche die Anfrage zutrifft
- Eine Liste mit den Schlüsseln der für die Anfrage relevanten Zellen, die nicht im Cache enthalten sind

Die Liste mit den Zellschlüsseln aus dem vorläufigen Ergebnis der Anfrage wird dann weiterverarbeitet, indem für jeden Nachbarknoten eine Liste von Zellschlüsseln erzeugt wird. In einem Druchlauf über die Liste der gesamten Zellschlüssel wird jeder Schlüssel zu der Liste des Nachbarns hinzugefügt, dessen Cache-Fokus am nähesten an der Position der Zelle liegt.

Dann wird gezählt, wieviele der Listen nicht leer sind, d.h. wieviele Nachbarn eine Teilanfrage erhalten sollen. Die Anzahl der Nachbarn wird ermittelt, damit der zentrale Knoten die Anfragebearbeitung abschließen kann, nachdem er von allen Nachbarknoten eine Antwort erhalten und die Teilantworten zusammengeführt hat.

An alle Nachbarn, die eine Teilanfrage erhalten sollen, wird eine PartialQuery Nachricht (s. Kapitel 4.3.1), mit der Anfrage, der Anzahl der Nachbarn, sowie den noch fehlenden Zellschlüsseln gesendet.

Ein Knoten, der eine PartialQuery Nachricht erhält, in der noch kein Teilergebnis gespeichert wurde, berechnet über das CacheProtocol seine Teilantwort, speichert sie in der PartialQuery Nachricht und schickt die Nachricht zurück zum Sender der Nachricht.

Wenn ein Knoten eine PartialQuery Nachricht mit einem Teilergebnis erhält, so wird das Teilergebnis zum vorläufigen Ergebnis der Anfrage im CacheProtocol hinzugefügt und der Zähler im vorläufigen Ergebnis hochgezählt (s. Kapitel 4.2.3). Wenn der Zähler mit der in der PartialQuery Nachricht gespeicherten Anzahl an Nachbarn übereinstimmt, ist die Zusammenführung der Teilergebnisse abgeschlossen.

Sind in dem vorläufigen Ergebnis alle Objekte zur Beantwortung der Anfrage enthalten, ist die Verarbeitung damit abgeschlossen und die Anfrage konnte vollständig aus dem Cache-Overlay beantwortet werden. Wenn in dem vorläufigen Ergebnis noch Zellen fehlen, wird eine Anfrage an das Daten-Back-End geschickt und das Ergebnis aus dem Daten-Back-End wird mit dem vorläufigen Ergebnis zum vollständigen Ergebnis zusammengeführt.

4.2.2 LAOLink

Das Protokoll LAOLink wurde in dieser Arbeit nicht verändert. Die Funktionen des Protokolls werden im Folgenden dargestellt, um die Zusammenarbeit der verschiedenen Protokolle zu verdeutlichen. Zudem wurden im Protokoll VoroLink (s. Kapitel 4.2.5) die Funktionen des Protokolls LAOLink übernommen.

Die Funktion des Protokolls besteht in der Speicherung und Verwaltung der Routing-Tabelle des Knotens. Das Protokoll bietet daher Funktionen zum Hinzufügen und Entfernen von Nachbarknoten, zur Ausgabe einer Liste der Nachbarknoten, sowie zur Ausgabe des Nachbarknotens mit der gerinsten Entfernung zu einem gegebenen Punkt.

4.2.3 CacheProtocol

In dem Protokoll CacheProtocol wird der Cache eines Knotes verwaltet. Die bereitgestellten Funktionen umfassen die Ausgabe der auf eine Anfrage zutreffenden Zellen im Cache, das Anpassen des CacheFokus, sowie die Ausgabe der Kapazität und der aktuellen Anfragelast des Cache.

Für die Umsetzung des kooperativen Verarbeitungsmodus (s. Kapitel 3.2.3, 4.2.1) wurde dem Protokoll eine Hash-Map für die Speicherung der Teilergebnisse der Anfragen hinzugefügt.

Für die Speicherung und Zusammenführung der Teilergebnisse wird der Datentyp QueryResult verwendet. Als Schlüssel für die Speicherung der QueryResults in der Hash-Map wird der Hash-Code der Query verwendet.

In einem QueryResult werden die aus dem Cache gelesenen Objekte und die für die Anfrage noch fehlenden Zellen, sowie der zentrale Knoten von dem die Anfrage verarbeitet wird gespeichert. Zudem enthält das QueryResult einen Counter, der hochgezählt wird, wenn das Ergebnis eines weiteren Knotens zum Ergebnis hinzugefügt wird.

4.2.4 VoroProtocol

Das Protkoll VoroProtocol bietet für die Voronoi-Delaunay Topologie zunächst einmal die gleichen Funktionen wie das Protokoll LAOProtocol für die Feder-Partikel Topologie. Allerdings wurde für die Aktualisierung der Knotenposition der in Kapitel 3.3.5 beschriebene Algorithmus implementiert, und es wurden entsprechende Funktionen für die Verarbeitung von Nachrichten zur Aktualisierung des Overlays und zum Hinzufügen neuer Knoten ergänzt.

Sowohl die Aktualisierung des Overlays als auch das Hinzufügen neuer Knoten wurde über einen Triggermechanismus implementiert, der in bestimmten Zeitabständen den Versand von MoveNodePoint- und NewNodePoint-Nachrichten anstößt. Die Zeitabstände können in der Konfigurationsdatei vorgegeben werden. Diese Vorgehen wurde nach dem

Vorbild des bereits implementierten Versand der Anfragen an das Netzwerk gewählt

Zur Aktualisierung des Overlays wird eine MoveNodePoint-Nachricht an einen zufällig ausgewählten Knoten gesendet. Ein Knoten, der eine MoveNodePoint-Nachricht erhält und sich zuvor selbst als bewegt markiert hat, bestimmt zunächst den Nachbarn, der am nähesten an der eigenen Position liegt. Dann entfernt er alle anderen Nachbarn aus seiner Routing-Tabelle und erstellt eine NewNodePoint-Nachricht mit seinem eigenen Knotenpunkt als Attribut. Diese NewNodePoint-Nachricht wird an den nähesten Nachbarn geschickt.

Für das Hinzufügen eines neuen Knotens werden zunächst Zufallskoordinaten im Koordinatenraum des Netzwerks erzeugt. Dann wird eine NewNodePoint-Nachricht erzeugt, die einen Knotenpunkt mit den Zufallskoordinaten, aber noch ohne Cache-Knoten enthält. Diese Nachricht wird in einer Route-Nachricht an einen zufällig ausgewählten Knoten geschickt.

Wie im LAOProtocol wird die Route-Nachricht an den Knoten weitergeleitet, der am nähesten an den Koordinaten liegt. Von diesem Knoten wird die Nachricht aus der Route-Nachricht extrahiert und verarbeitet.

Bei der Verarbeitung der Nachricht wird ein neuer Cache-Knoten zum Netzwerk hinzugefügt. Der verarbeitende Knoten trägt den neuen Knoten in seiner Routing-Tabelle ein und in der Routing-Tabelle des neuen Knotens wird der verabeitende Knoten eingetragen. Von diesem Punkt an ist der neue Knoten ein Teil des Netzwerks. Der verarbeitende Knoten setzt die Variable "first" in der NewNodePoint-Nachricht auf false und leitet die Nachricht an seine Nachbarn weiter.

Erhält ein Knoten eine NewNodePoint-Nachricht, deren Variable "first" auf false gesetzt wurde, so aktualisiert er wie in Kapitel 3.3.4 beschrieben seine Routing-Tabelle und leitet die Nachricht ggf. an seine Nachbarn weiter. Jeder Knoten, der den neuen Knoten zu seiner Routing-Tabelle hinzufügt, wird auch zu der Routing-Tabelle des neuen Knotens hinzugefügt.

4.2.5 VoroLink

Das Protokoll VoroLink enthält die gleichen Funktionen und Komponenten wie das Protokoll LAOLink und zudem noch eine Triangulierung über die Nachbarknoten und Funktionen zur Aktualisierung und Ausgabe der Triangulierung.

Da die Triangulierung mit dem Computational Geometry Paket aus [Baro8] erzeugt wird, sind die Funktionen zur Aktualisierung und Ausgabe lediglich Wrapper-Funktionen, die auf Funktionen aus dem Paket zugreifen.

4.3 Nachrichten

4.3.1 PartialQuery

Die Nachricht PartialQuery wird für die Anfrageverarbeitung im kooperativen Modus verwendet. Sie wird vom zentralen Knoten im Laufe der Anfrageverarbeitung an die Nachbarknoten versandt und von den Nachbarknoten mit entsprechenden Teilergebnissen zurückgeschickt.

Eine PartialQuery Nachricht enthält die folgenden Komponenten:

- die Anfrage
- die Anzahl der Nachbarn, an die eine Teilanfrage der Ursprungsanfrage geschickt wurde
- die Schlüssel der noch fehlenden Zellen
- das Ergebnis der Teilanfrage

4.3.2 NewNodePoint

Die Nachricht NewNodePoint hat zwei Verwendungen. Zum einen wird diese Nachricht verwendet, um neue Knotenpunkte zum Netzwerk hinzuzufügen und zum anderen wird bei der Aktualisierung der Nachbarschaftsbeziehungen ebenfalls eine NewNodePoint-Nachricht verwendet.

Eine NewNodePoint-Nachricht enthält einen Knotenpunkt. Wenn die Nachricht zum Hinzufügen eines neuen Knotenpunktes verwendet wird, enthält der Knotenpunkt zunächst keinen Cache-Knoten. Dieser wird von dem ersten Knoten, der die Nachricht verarbeitet zum Netzwerk hinzugefügt.

Wird die Nachricht zur Aktualisierung verwendet, dann entspricht der in der Nachricht enthaltene Knotenpunkt dem Knoten, dessen Nachbarschaftsbeziehungen aktualisiert werden sollen.

4.3.3 MoveNodePoint

Die Nachricht MoveNodePoint wird zur Aktualisierung der Nachbarschaftsbeziehungen im Netzwerk verwendet. In einem in der Konfigurationsdatei vorgegebenen Zeitintervall wird einem zufällig ausgewählten Knoten im Netzwerk eine MoveNodePoint-Nachricht geschickt. Durch diese Nachricht wird im Protokoll VoroProtocol die Aktualisierung der Nachbarschaftsbeziehungen dieses Knotens angestoßen.

5 Zusammenfassung und Ausblick

Tabelle 5.1 bietet eine Übersicht über den Eigenanteil an den in dieser Arbeit beschriebenen Komponenten.

Die in Kapitel 3.1 vorgestellte Topologie wurde in [LRM12] ausgewertet und bietet eine schnelle Anpassung an sich veränderde Anfragehotspots, ohne das die Topologie des Overlays aktualisiert werden muss. Für das Hinzufügen neuer Knoten zum Netzwerk sind wie in [LRM12] beschrieben, besondere Maßnahmen zur Erhaltung der Stabilität des Netzwerks nötig.

Für die in Kapitel 3.3 vorgestellte Topologie basierend auf einer Schwerpunkt-Voronoi-Delaunay-Triangulierung sind Aktualisierungen der Struktur des Cache-Overlays nötig. Dafür bietet diese Topologie im Vergleich zur Feder-Partikel-Topologie jedoch eine größere Flexibilität in der Positionierung der Knoten, sowie für das Hinzufügen neuer Knoten zum Netzwerk.

In der Voronoi-Delaunay-Topologie kann die Initialisierung der Knotenpositionen auch basierend auf Kriterien für prognostizierbare Hotspots erfolgen. So kann zum Beispiel die Bevölkerungsdichte ein Kriterium für die abzusehende Menge an Anfragen sein, da z.B. in Großstädten mit hoher Bevölkerungsdichte auch mit einer größeren Anfragedichte zu rechnen ist als in ländlichen Regionen.

Auch das Hinzufügen eines neuen Knotens ins Netzwerks könnte gezielt in Hotspot-Regionen erfolgen, indem ein neuer Knoten zunächst an einen Knoten mit hoher Last weitergeleitet und dann in der Nähe dieses Knotens positioniert wird.

Die in Kapitel 3.3.5 beschriebene Dichtefunktion bietet ebenfalls noch Potential zu einer besseren Anpassung des Netzwerks an Anfragehotspots. Durch die Weiterentwicklung der noch recht einfach gehaltenen Funktion

Komponente	In dieser Arbeit erstellt
Klasse SimpleTopol	Topologie Diagonal
Klasse VoroTopol	Topologie Voronoi-Delaunay
Protokoll LAOProtocol	Modus Niedrigste Last Kooperativer Modus
Protokoll CacheProtocol	Hash-Map zur Speicherung der Teilergebnisse von Anfragen Datentyp QueryResult
Protokoll VoroProtocol	Implementierung des K-Means Algorithmus Aktualisierung des Overlays Hinzufügen neuer Knoten
Protokoll VoroLink	Triangulierung über Nachbarknoten Aktualisierung der Triangulierung Anpassen der Routing-Tabelle anhand der Triangulierung
Nachrichten	PartialQuery NewNodePoint MoveNodePoint

Tabelle 5.1: Zusammenfassung des Eigenanteils an den in dieser Arbeit beschriebenen Komponenten

könnten die Anpassungsfähigkeit des Netzwerks und die Verteilung der Last wahrscheinlich verbessert werden.

Literaturverzeichnis

[Aur91] F. Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991. doi:10.1145/116873.116880. URL http://doi.acm.org/10.1145/116873.116880. (Zitiert auf Seite 24)

[Bar08] M. Bartoletti. Computational Geometry Tutorial, 2000-2008. URL http://cgtutorial.sourceforge.net/. (Zitiert auf den Seiten 29, 38 und 44)

[DFG99] Q. Du, V. Faber, M. Gunzburger. Centroidal Voronoi Tessellations: Applications and Algorithms. *SIAM Rev.*, 41(4):637–676, 1999. doi:10.1137/S0036144599352836. URL http://dx.doi.org/10.1137/S0036144599352836. (Zitiert auf Seite 24)

[DFJ+96] S. Dar, M. J. Franklin, B. T. Jonsson, D. Srivastava, M. Tan. Semantic Data Caching and Replacement, 1996. (Zitiert auf Seite 10)

[DG02] Q. Du, M. Gunzburger. Grid generation and optimization based on centroidal Voronoi tessellations. *Appl. Math. Comput.*, 133(2-3):591–607, 2002. doi:10.1016/S0096-3003(01)00260-0. URL http://dx.doi.org/10.1016/S0096-3003(01)00260-0. (Zitiert auf den Seiten 24, 26 und 27)

[JMJV10] M. Jelasity, A. Montresor, G. P. Jesi, S. Voulgaris. Peersim, 2010. URL http://peersim.sourceforge.net/. (Zitiert auf Seite 35)

[Kle89] R. Klein. *Concrete and Abstract Voronoi Diagrams*, Band 400 von *Lecture Notes in Computer Science*. Springer, 1989. (Zitiert auf Seite 24)

[Lae] K. V. Laerhoven. Voronoi Diagrams and Delaunay Triangulation. URL http://www.comp.lancs.ac.uk/~kristof/research/notes/voronoi/. (Zitiert auf Seite 25)

[LBC+11] C. Lübbe, A. Brodt, N. Cipriani, M. Grossandmann, B. Mitschang. DiSCO: A Distributed Semantic Cache Overlay for Location-Based Services. In *Mobile Data Management (MDM), 2011 12th IEEE International Conference on*, Band 1, S. 17 –26. 2011. (Zitiert auf Seite 9)

[LRM12] C. Lübbe, A. Reuter, B. Mitschang. Elastic Load-Balancing in a Distributed Spatial Cache Overlay. In *Proc. of the 13th International Conference on Mobile Data Management (MDM)*. IEEE Computer Society, Washington, DC, USA, 2012. To appear. (Zitiert auf den Seiten 7, 8, 10, 11, 16, 17, 18, 35 und 47)

[Mac67] J. B. MacQueen. Some Methods for Classification and Analysis of MultiVariate Observations. In L. M. L. Cam, J. Neyman, Herausgeber, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, Band 1, S. 281–297. University of California Press, 1967. (Zitiert auf Seite 27)

[NM04] D. Nicklas, B. Mitschang. On building location aware applications using an open platform based on the Nexus Augmented World Model. S. 303–313, 2004. (Zitiert auf Seite 9)

[OBSC00] A. Okabe, B. Boots, K. Sugihara, S. N. Chiu. *Spatial tessellations: Concepts and applications of Voronoi diagrams*. Probability and Statistics. Wiley, NYC, 2nd Auflage, 2000. 671 pages. (Zitiert auf Seite 24)

Erklärung

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

(Anja Reuter)