Optimized Information Discovery in Structured Peer-to-Peer Overlay Networks

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Faraz Ahmed Memon

aus Karachi, Pakistan

Hauptberichter:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Mitberichter:	Prof. Dr. rer. nat. Winfried Lamersdorf
Tag der mündlichen Prüfung:	8. Juli 2011

Institut für Parallele und Verteilte Systeme (IPVS)
der Universität Stuttgart
2011

Acknowledgments

First and foremost, thanks to almighty God for giving me the strength, the courage, and the patience for achieving this important milestone in my life. It has been a humbling experience.

I would like to thank Prof. Dr. Kurt Rothermel for recognizing the potential in me and selecting me for conducting research under his supervision. His guidance played an essential role in recognition of research problems and development of their solutions during my doctoral studies. Moreover, I would like to thank Prof. Dr. Winfied Lamersdorf for taking the time to review my dissertation.

I dedicate this dissertation to: my parents, my wife, my son, and my siblings. Living in a country with extreme poverty and high illiteracy, my father, Taj Mohammad Memon, worked day and night to provide me food, shelter, and the best education in the world. My mother, Naseem Taj Memon, taught me the moral and the social values in life, making me the person I am today. She supported me through each step of the earlier years of my education. I remember vividly, the times when she would stay up all night just to see if I was hungry while studying for exams. She prayed for my success more than I did, and she still does that today. My wife, Faryal Gul Memon, has been my companion only since few years, but I feel that she knows me since a lifetime. She bore with me through the years of my doctoral study and she is one of the most understanding persons that I know. My son, Eyad Faraz Memon, has given a new meaning to my life. Being a father has made me more responsible and caring person than I ever was. My siblings, Faisal Taj Memon, Huda Taj Memon, and Talha Taj Memon, have always been there to share the happy and the sad moments of my life.

Apart from my family, continual moral and technical support of several individuals has made this dissertation possible. I am inclined to name some of these wonderful individuals below.

Dr. Frank Dürr, my project supervisor, has been very supportive throughout my doctoral studies. He has always been available for discussions and has given me time under busy situations where anyone would have excused. Whether it was a research dialogue, a programming tip, a documentation hint, or a resource allocation problem, Frank was always there to facilitate the need.

Through the years of my doctoral studies, I have had the opportunity to work with some wonderful people as colleagues. The support that I received from the following people has made me think of them as friends rather than just co-workers. Steffen Maier, guided me during the first year of my doctoral studies to facilitate my integration into the research world. Brilliant programming skills of Daniel Tiebler were extremely handy when it came to development of prototype solutions to the research problems. Gerhald Koch, Stefan Föll, and Bilal Hameed, provided their valuable feedbacks on the research papers that I submitted for review at the international conferences. Christian Hiesinger, Lars Geiger, Manuel Gonzalo, and Adnan Tariq, always welcomed me whenever I felt the need for discussing ideas and research problems. Stephan Schuhmann, Andreas Benzing, Marco Völz, and Stamatia Rizou, helped me with non-technical and administrative issues during my doctoral studies.

To all the people mentioned above, and to the unmentioned ones who helped me in any way, I am thankful from the bottom of my heart.

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

P2P	Peer-to-Peer
IP	Internet Protocol
DHT	Distributed Hash Table
DBMS	Database Management Systems
SFC	Space-filling Curve
OID	Optimized Information Discovery
NFS	Network File System
LAN	Local Area Network
DSL	Digital Subscriber Line
BFS	Breadth-first Search
TTL	Time to Live
DFS	Depth-first Search
APS	Adaptive Probabilistic Search
ARPS	Adaptive Resource-based Probabilistic Search
HPF	Hybrid Periodic Flooding
RIs	Routing Indices
SHA	Secure Hash Algorithm
CAN	Content Addressable Network

Abstract

Peer-to-peer (P2P) overlay networks allow for efficient information discovery in large-scale distributed systems. Although point queries are well supported by current P2P systems – in particular systems based on distributed hash tables (DHTs) –, providing efficient support for more complex queries remains a challenge. Therefore, the goal of this research is to develop methodologies that enable efficient processing of complex queries, in particular processing of multi-attribute range queries, over DHTs.

Generally, the support for multi-attribute range queries over DHTs has been provided either by creating an individual index for each data attribute or by creating a single index using the combination of all data attributes. In contrast to these approaches, we propose to create and modify indices using the attribute combinations that dynamically appear in multi-attribute range queries in the system.

In order to limit the overhead induced by index maintenance, the total number of created indices has to be limited. Thus, one of the major problems is to create a limited number of indices such that the overall system performance is optimal for multi-attribute range queries. We propose several index recommendation algorithms that implement heuristic solutions to this NP-hard problem. Our evaluations show that these heuristics lead to a close-to-optimal system performance for multi-attribute range queries.

The final outcome of this research is an adaptive DHT-based information discovery system that adapts its set of indices according to the dynamic load of multi-attribute range queries in the system. The index adaptation is carried out using a four-phase index adaptation process. Our evaluations show that the adaptive information discovery system continuously optimizes the overall system performance for multi-attribute range queries. Moreover, compared to a non-adaptive system, our system achieves several orders of a magnitude improved performance.

Zusammenfassung

Einleitung

Moderne Peer-to-Peer-Anwendungen – z.B. zum Auffinden von Speicher- und Rechenressourcen (engl. Resource Discovery) im Grid-Computing [MRPM08], Peer-to-Peer-Video-Streaming-Systeme [ND08] oder P2P-Systeme zur Verwaltung geographischer Informationen (Spatial Information Discovery) [MTD$^+$09] – erfordern Unterstützung für mehrdimensionale Bereichsanfragen.

Existierende unstrukturierte Peer-to-Peer-Netze unterstützen zwar eine große Anzahl an Anfragetypen (einschließlich mehrdimensionaler Bereichsanfragen), bieten jedoch keine Garantien bezüglich des erfolgreichen Auffindens von Informationen. Deshalb wurden strukturierte Peer-to-Peer-Netze, insbesondere so genannte Verteilte Hash-Tabellen (engl. Distributed Hash Table, kurz DHT), zur Unterstützung mehrdimensionaler Bereichsanfragen erweitert. Im Allgemeinen werden hierzu zusätzliche Indexierungsmechanismen eingeführt, welche als zusätzliche Schicht oberhalb der DHT realisiert sind. In der Literatur werden hierfür die beiden folgenden Indexierungsansätze diskutiert.

Beim ersten Ansatz wird eine Indexstruktur für *jedes einzelne Attribut* eines Datenobjekts erstellt [AX02, CFCS03, SOTZ05, TP03]. Bei dieser Vorgehensweise wird eine multidimensionale Bereichsanfrage auf eine der beiden folgenden Weisen abgearbeitet: Entweder wird die gesamte Anfrage mit nur einem Attribut mit Hilfe einer DHT zu den entsprechenden Peers weitergeleitet, die dann eine Filterung anhand der verbleibenden Attribute durchführen. Oder die Anfrage wird in mehrere Bereichsanfragen aufgeteilt, so dass jede Anfrage ein Attribut der ursprünglich mehrdimensionalen Anfrage enthält. Diese Anfragen werden als eindimensionale Bereichsanfragen separat durchgeführt, und die resultierden Ergebnisse werden beim anfragenden Peer zusammengeführt und gefiltert. Das erste Virgehen hat dabei den Nachteil, dass sich zum einen die Latenz durch die notwendige Filterung bei den Peers erhöht. Zum anderen führt dieser Ansatz zu einer erhöhten Netzlast, da die Vermittlung der Anfrage nur anhand eines Attributs der ursprünglichen Anfrage erfolgt und somit auch Peers die Anfrage erhalten, die nicht alle Bereiche der mehrdimensionalen Anfrage abdecken. Beim zweiten Vorgehen wird eine große Anzahl von Datenobjekten an den anfragenden Peer übertragen, wobei nur einige in den Bereichen aller ursprünglich angefragten Attribute liegen.

Beim zweiten Ansatz wird eine *einzige Indexstruktur* erstellt, die eine Kombination aller Attribute darstellt [CRR+05, GYGM04, SP03]. Eine mehrdimensionale Anfrage kann hierbei im Gegensatz zum ersten Ansatz direkt nur an diejenigen Peers gesendet werden, die für die Abarbeitung tatsächlich zuständig sind. Für Attribute, die nicht in der Anfrage vorkommen, müssen Platzhalter (engl. Wildcard) für alle möglichen Attributwerte angenommen werden. Mit steigender Anzahl der Platzhalter sinkt dabei die Leistung dieses Ansatzes bzw. steigt die Netzlast.

Zur Verbesserung dieser Nachteile wird in dieser Dissertation das so genannte *Optimized-Information-Discovery-System* (kurz *OID*) vorgeschlagen, das eine dritte Art der Indexierung von Datenobjekten verwendet. Bei einer vorgegebenen Menge von Attributen erstellt das OID-System mehrere Indexstrukturen, bei der jede aus einer Kombination bestimmter (nicht notwendigerweise aller) Attribute besteht. Diese Attributkombinationen können derart gewählt werden, dass die Leistung des Systems bei mehrdimensionalen Bereichsanfragen für eine gewisse Anfragemenge optimal ist. Darüber hinaus stellen wir ein Verfahren vor, das für hochskalierbare, verteilte Anwendungen die Indexmenge im laufenden Betrieb adaptiert, um auch bei sich ändernden Anfragen eine optimale Leistung zur ermöglichen. Die Effektivität und Effizienz des Systems wurde dabei durch eine Implementierung des Systems für ein Simulationswerkzeug und verschiedene Simulationen nachgewiesen.

Grundlagen

Zu Beginn der Computernetzära waren Client-Server-Architekturen verbreitet. Aufgrund des technologischen Fortschritts, der zu immer leistungsfähigeren Rechnern und breitbandigen Internetzugängen führte, die Privatanwendern zu günstigen Tarifen z.B. als Flatrates angeboten werden, erfreuen sich allerdings Peer-to-Peer-basierte Netze zunehmend einer großen Beliebtheit. Eine solches Peer-to-Peer-System besteht dabei aus mehreren Computern (Peers), die miteinander verbunden sind und ein sogenanntes Overlay-Netz auf Basis der IP-Netzinfrastruktur bilden. Im Gegensatz zu Client/Server-Systemem nutzt ein Peer nutzt nicht nur einen Dienst, sondern bietet ebenfalls den anderen Peers diesen Dienst an, das heißt, ein Peer agiert sowohl als Klient als auch als Server. Peer-to-Peer-Systeme werden dabei in zwei Kategorien eingeteilt: unstrukturierte und strukturierte Peer-to-Peer-Systeme.

In einem unstrukturierten Peer-to-Peer-System ist die Topologie des Netzes unabhängig von den Daten, die ein Peer verwaltet, das heißt, das Netz bestimmt nicht den Speicherort der Daten. Vielmehr werden Verbindungen zwischen Peers oft zufällig gewählt, woraus sich meist ein sogenanntes Small-World-Netz [MN04] ergibt. Aufgrund einer fehlenden strikten Organisation des Netztes werden meist vergleichsweise einfache Vermittlungsalgorithmen basierende z.B. auf der Breiten- oder Tiefensuche, Random-Walker, o.ä. eingesetzt. Solche Netze unterstützen sowohl einfache als auch komplexe Suchanfragen. Allerdings basiert die Suche in diesen Netzen auf einem Best-Effort-Ansatz, das heißt, es gibt keine Garantien, dass die Suche erfolgreich ist, selbst wenn die Daten im Netz existieren.

Im Unterschied zu unstrukturierten P2P-Systemen werden die Daten in strukturierten Netzen von genau bestimmten, festgelegten Peers Netz verwaltet. Die Netztopologie folgt strikten Regeln, so dass Suchanfragen für bestimmte Daten gezielt in die Richtung der Peers weitergeleitet werden können, welcher diese Daten verwaltet.

Ein strukturiertes Peer-to-Peer-System wird oft als eine so genannte Verteilte Hash-Tabelle (engl. Distributed Hash Table; DHT) implementiert. Das OID-System verwendete z.B. die Chord DHT [SMK+01]. Entsprechen der Funktionalität einer DHT ermöglicht Chord das Auffinden eines mit einem Hash-Schlüssel indizierten Datenobjekts. Chord erstellt hierzu ein Netz aus Peers, die in einem Ring angeordnet sind. Der Ring wird dabei auf Grundlage von eindimensionalen numerischen Bezeichnern (modulo 2^m) bestimmt. Jedem Peer wird dabei ein m Bit großer Bezeichner aus dem Bezeichnerbereich $[0, 2^m)$ zugewiesen. Dieser wird erzeugt durch die Anwendung einer Hash-Funktion, wie beispielsweise SHA-1, die z.B. auf die IP-Adresse des Peer angewandt wird. Das Chord-Protokoll stellt die Funktionen *lookup(key)* und *notify()* bereit. Die Funktion *lookup(key)* gibt die IP-Adresse des Peers zurück, dessen Bezeichner dem Bezeichner des Schlüssels im Bezeichnerring gleich ist oder diesem folgt. Dieser Peer wird als Nachfolger des Schlüssels bezeichnet. Die Funktion *notify()* benachrichtigt über eine Änderung in der Schlüsselmenge eines Peers.

Für eine effiziente Vermittlungs von Suchanfragen verwaltet ein Chord-Peer m Verbindungen zu anderen Peers in seiner Vermittlungstabelle, der sogenannten Fingertabelle. Ein Peer mit dem Bezeichner n besitzt als i-ten Eintrag in seiner Fingertabelle den Bezeichner des Peers, welcher der Nachfolger des Bezeichners $n + 2^{i-1}$ auf dem Bezeichnerring ist. Die Fingertabelle garantiert, dass eine Suchanfrage in $O(logN)$ Vermittlungsschritten (engl. Hop) im Overlay-Netz bis zum Ziel-Peer weitergeleitet wird, wobei N die Anzahl der Peers im Netzwerk ist.

DHTs wie Chord sind aufgrund des eindimensionalen Bezeichnerraums nur in der Lage 1-dimensionale Daten, die aus einem Schlüssel-Wert-Paar bestehen, zu speichern und zu suchen. Jedoch verwenden anspruchsvolle Anwendungen, wie beispielsweise geographische Informationssysteme, Dokumentenverwaltungssysteme oder Ressourcenmanagementsysteme für gewöhnlich Datenobjekte mit mehreren Attributen (Schlüssel-Wert-Paaren). Solche Anwendungen benötigen üblicherweise eine effiziente Umsetzung von Suchanfragen über mehrere Attribute und Wertebereiche. Zum selben Zweck verwenden Datenbankmanagementsysteme (DBMS) Indexstrukturen, die Daten in einem Speicher derart organisieren, dass auf diese schnell und effizient zugegriffen werden kann. Eine solche Indexstruktur ist eine sogenannten raumfüllende Kurve (space filling curve, SFC). Unser System bedient sich der SFC von Hilbert [Hil91], die sogenannte Hilbert-Kurve, um Daten in einem DHT-basierten Peer-to-Peer-System zu indexieren.

Die Konstruktion einer Hilbert-Kurve in einem d-dimensionalen Raum erfolgt in zwei Schritten. Im ersten Schritt wird der Raum in kleinere Unterräume unterteilt, welche Zonen genannt werden. Diese Unterteilung kann als rekursiver Vorgang betrachtet werden, wobei der Raum mit jedem Durchgang in 2^d Zonen unterteilt wird. Dies wird k-mal durchgeführt und ergibt $2^{k \cdot d}$ Zonen, wobei k als Annäherungsstufe (engl. Approximation Level) bezeichnet wird. Im zweiten Schritt wird eine Linie definiert, die einmal durch jede der $2^{k \cdot d}$ Zonen geht und die Zentren von zwei Zonen mit einem Liniensegment verbindet. Das Zentrum einer Zone ist eine Näherung für alle Punkte in der Zone, und die Verbindungslinie zwischen zwei benachbarten Zonen stellt eine Ordnung zwischen diesen her. Die resultierende Kurve ist eine raumfüllende Kurve der k-ten Ordnung in einem d-dimensionalen Raum.

Architektur des OID-Systems

Das OID-System besteht aus einer dreischichtigen Architektur (siehe Abbildung 1 unten). Die oberste Schicht ist die Anwendungsschicht, die aus einer Menge verteilter Anwendungen besteht, welche die Funktionalität von mehrdimensionalen Bereichsanfragen benötigen. Die mittlere Schicht besteht aus dem OID-Rahmenwerk, das selbst wiederum klassifiziert werden kann in Datenindexraum, Datenplatzierungs-Controller und Anfragekomponente.

Die unterste Schicht ist die DHT-Schicht. OID nimmt eine DHT an, die unter Verwendung der Funktion $lookup(key)$ denjenigen Peer bestimmt, welcher für den Suchschlüssel verantwortlich ist. Ferner werden alle Änderungen im Netz über die Rückruffunktion $notify()$ gemeldet.

Ein Datenobjekt wird im OID-System durch eine Liste von Attribut-Wert-Paaren repräsentiert. Wird jedes Attribut eines Datenobjekts als eine Dimension und die Kombination aus Attributwerten als Koordinaten betrachtet, so kann jedes Datenobjekt als ein Punkt in einem mehrdimensionalen Raum gesehen werden. Die Abbildung eines mehrdimensionalen Punkts auf einen eindimensionalen Bezeichner erfolgt mit Hilfe raumfüllender Kurven.

Der Datenindexraum der OID-Rahmenwerksschicht beinhaltet dabei o benutzerdefinierte, SFC-basierte Indizes. Diese Indizes werden vom Designer der verteilten Anwendung definiert, wobei diejenigen Attributkombinationen verwendet werden, welche in den populärsten Suchanfragen vorkommen. Auf diese Weise werden immer die populärsten Suchanfragen im System optimiert, was zu einer Optimierung der Leistung des Gesamtsystems führt.

Die Datenplatzierungs-Controller benutzt die Indices des Datenindexraums, um den Datenobjekten Bezeichner zuzuweisen. Daraufhin wird jedes Datenobjekt zu demjeningen

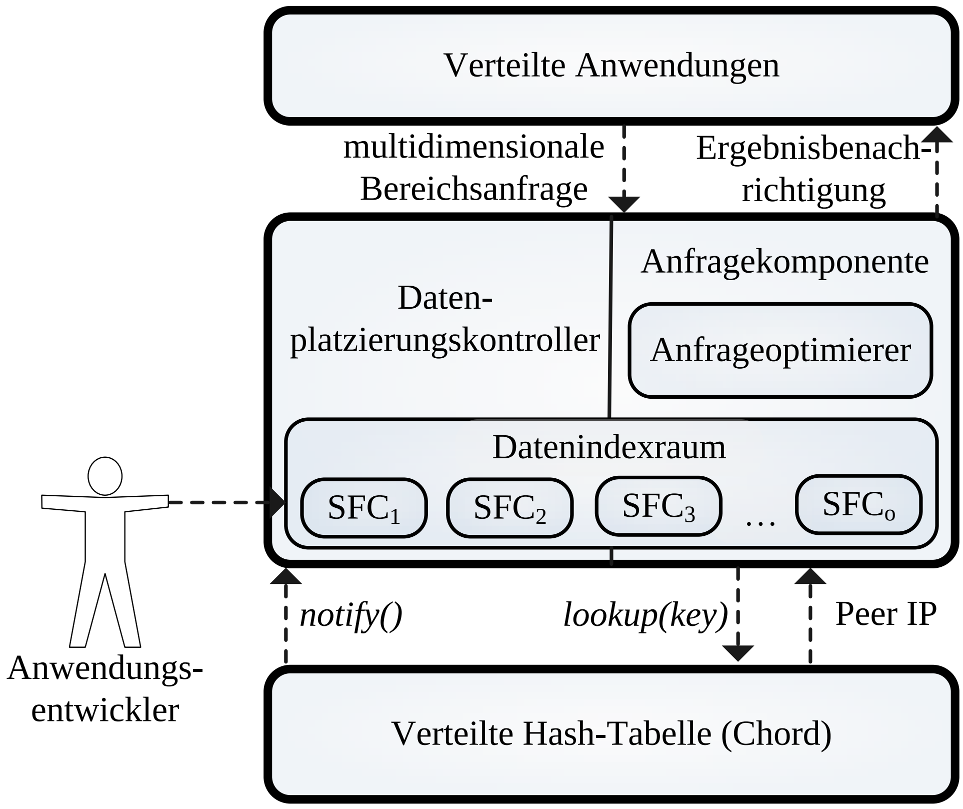


Abbildung 1: OID Systemmodell

Peer weitergeleitet, der für dessen Bezeichner verantwortlich ist. Da der Bezeichnerraum einer SFC normalerweise kleiner ist als der Bezeichnerraum einer DHT, wird der Bezeichner des Datenobjekts mittels Skalierung auf den Bezeichnerraum der DHT abgebildet, bevor der verantwortliche Peer ermittelt wird, welcher das Datenobjekt verwaltet.

In unserem System sind Suchanfragen als Konjunktion von Tupeln der Form (attribute *operator* value) definiert, wobei folgende Operatoren unterstützt werden: $=, \neq, <, >, \leq$ and $\geq$. Der Anfrageoptimierer ist für die Abarbeitung der Suchanfragen zuständig. Erhält dieser eine mehrdimensionale Bereichsanfrage, so wird die Anfrage zuerst auf alle SFC-basierten Indices des Datenindexraums abgebildet. Entweder stimmt die Anfrage vollständig mit einem der Indices bezüglich der angefragten Attribute überein, oder es handelt sich um eine teilweise Übereinstimmung mit mehr als einem Index. Falls es eine vollständige Übereinstimmung der Anfrage mit einem Index gibt, dann wird dieser Index für die weitere Abarbeitung der Suchanfrage verwendet. Anderenfalls wählt der Anfrageoptimierer aus den Indices, die nur teilweise übereinstimmen, denjenigen aus, der für die Abarbeitung der Anfrage der effizienteste ist. Die Effizienz eines teilweise übereinstimmenden Indexes wird bestimmt, indem geschätzt wird, an wie viele Peers die Anfrage gesendet werden müsste.

Sobald der effizienteste SFC-basierte Index bestimmt wurde, verfeinert der Anfrageoptimierer die Suchanfrage mittels dieses Indexes, um die entsprechenden Bezeichner der DHT bzw. der zuständigen Peers zu erhalten, an die die Anfrage weiterzuleiten ist. Eine naive Strategie würde nun für die ermittelten Bezeichner durch Aufruf der DHT-Funktion lookup(key) die zuständigen Peers ermitteln. Jedoch würde diese Strategie eine große Netzlast erzeugen und wäre somit nicht skalierbar. Aus diesem Grund wurde in dieser Arbeit eine optimierte, baumbasierte Vermittlung vorgeschlagen, bei der eine geringere Nachrichtenlast auf Kosten einen erhöhten Suchlatenz (Pfadlänge) erzielt wird, um eine skalierbare Abarbeitung der Suchanfragen zu erreichen. Die Verfeinerung einer mehrdimensionalen Bereichsanfrage mittels eines SFC-basierten Indexes kann beträchtliche Zeit in Anspruch nehmen. Deshalb wurde des Weiteren eine Strategie zur Verteilung der Berechnungslast vor, die es erlaubt, eine Suchanfrage parallel auf einer großen Anzahl von Peers in der DHT verfeinern zu lassen.

Die Evaluierung des OID-Systems zeigte, dass die beste Performance für mehrdimensionale Bereichsanfragen mit vollständig übereinstimmenden, SFC-basierten Indices erreicht wird. Deshalb ist ein einzelner, SFC-basierter Index als Kombination aller Attribute nicht ausreichend, falls häufig nur eine Teil der Attribute für Anfragen genutzt werden. Darüber hinaus konnten wir zeigen, dass der Algorithmus zur Optimierung der Vermittlung die Gesamtzahl der parallelen Nachrichten in Netz auf Kosten einer leicht erhöhten Latenz reduziert. Es wurde ebenfalls gezeigt, dass der Algorithmus zur Verteilung der Berechnungslast die Menge an Berechnungen, die ein einzelner Peer aufbringen muss, effektiv verringert.

Empfehlung von Indexstrukturen für die optimierte Suche nach Informationen

Wie oben beschrieben, basiert die Entscheidung für die initiale Indexmenge im OID-Rahmenwerk auf einer Heuristik, die eine vom Nutzer definierte Anzahl von Indices an Hand der populärsten Anfragen im System bestimmt. Dieses Vorgehen ist optimal für Systeme, in denen wenige populäre Anfragen den größten Anteil an der gesamten Anfragemenge ausmachen. Für Peer-to-Peer-Anwendungen mit wenigen populären Anfragen und einer grossen Menge an selteneren Anfragen führt dieses Vorgehen jedoch nur zu einer sub-optimal Indexmenge.

Deshalb stellen wir nachfolgend eine Reihe von Algorithmen vor, die Empfehlungen für die Suche in DHT-basierten Informationsverwaltungssystemen liefern. Die Algorithmen erhalten als Eingabe eine Schranke o für die maximale Anzahl an Indices sowie die Menge W an mehrdimensionalen Bereichsanfragen (engl. *Workload*), die vorher im laufenden System aufgezeichnet wurden. Basierend auf dieser Eingabe berechnet jeder Algorithmus eine Menge von Indices, die eine nahezu optimale Leistung des Systems für den Workload unter Berücksichtigung der Schranke für die maximale Anzahl an

Indices erreicht. Dabei wird davon ausgegangen, dass eine anwendungsbezogene Menge von Anfragen zur Verfügung steht.

Sei A der Bezeichner für die Menge der Attribute, die in den mehrdimensionalen Bereichsanfragen vorkommen. Eine naive Art, die optimale Indexmenge für einen Workload zu bestimmen, ist es, Kombinationen aus o Elementen der Potenzmenge von A aufzuzählen. Anschließend wählt man die Kombination mit den geringsten Kosten für die Anfragen in W aus. Wegen der verteilten Informationshaltung innerhalb des Peer-to-Peer-Systems können die tatsächlichen Kosten der Anfragen in W nur geschätzt werden. Zu diesem Zweck stellen wir eine Kostenfunktion vor, die auf der SFC-Indexstruktur basiert und aus zwei unterschiedlichen Indexmengen für eine bestimmte Anfrage diejenige Menge bestimmt, die die Anfrage effizienter verarbeitet.

Da durch die Potenzmenge die mögliche Anzahl von Lösungen exponentiell mit der Größe von A wächst, ist der naive Ansatz jedoch nicht skalierbar. Daher schlagen wir drei skalierbare Algorithmen zur Empfehlung von Indexstrukturen vor, die von Heuristiken Gebrauch machen.

Jeder der skalierbaren Algorithmen bestimmt zunächst ein Menge C von möglichen Indizes mit Hilfe der eindeutigen Attributkombinationen aus der Anfragemenge. Diese Menge von Kandidaten ist normalerweise größer als die nutzergegebene Schranke o für die maximale Anzahl der Indices. Deshalb wird diese Menge sukzessive weiter reduziert, indem einer der folgenden Algorithmen ausgeführt wird:

- **Kostenbasierte Vereinigung** – Der kostenbasierte Vereinigungsalgorithmus empfiehlt eine Menge von Indices, indem Paare von Attributkombinationen aus der Menge C von Kandidaten miteinander vereinigt werden, bis die Menge C eine Größe von $o - 1$ erreicht hat. Da die Vereinigung beliebiger Paare von Kombinationen aus C die Gesamtkosten stark erhöhen würde, werden nur die Paare miteinander vereinigt, die den geringsten Kostenzuwachs produzieren.
- **Ähnlichkeitsbasierte Vereinigung** – Der ähnlichkeitsbasierte Vereinigungsalgorithmus vereinigt ebenfalls Elemente aus C, um eine Indexmenge zu empfehlen. Dazu werden Paare, die hinsichtlich der enthaltenen Attribute am ähnlichsten zueinander sind, miteinander verschmolzen bis die Menge C auf die Größe $o - 1$ reduziert werden konnte. Im Vergleich zum kostenbasierten Vereinigungsalgorithmus hat dieser Ansatz einen geringeren Aufwand, weil die Elemente aus C vereinigt werden, ohne die direkten Kosten für den Workload zu betrachten. Der ähnlichkeitsbasierte Vereinigungsalgorithmus baisert auf der Beobachtung, dass durch die Vereinigung zweier fast identischer Indices die Performance des Systems nicht erheblich verschlechtert wird.
- **Auswahlalgorithmus** – Der Auswahlalgorithmus berechnet die Kosten für jedes Element der Menge C der Kandidaten und wählt die $o - 1$ Elemente mit den geringsten Kosten aus. Die Idee bei diesem Vorgehen ist, dass die Wahrscheinlich-

keit hoch ist, dass die Kosten für die gesamte Anfragemenge gering ist, wenn die Elemente einzeln für sich die geringsten Kosten aufweisen.

Jeder der oben besprochenen Indexempfehlungsalgorithmen reduziert die Größe der Menge von Kandidaten C auf $o-1$, damit noch ein weiterer Index aus der Kombination aller Attribute aus A miteinbezogen werden kann. Dieser Index kann für Anfragen eingesetzt werden, deren Ausführung durch die ausgewählten $o-1$ Indices unzureichend optimiert wurde.

Die Evaluierung der Indexempfehlungsalgorithmen zeigte, dass ein Kompromiss zwischen der Effizienz der ausgewählten Indices und der Ausführungszeit der Indexauswahl getroffen werden muss. Diejenigen Algorithmen, welche die effizientesten Indexstrukturen berechnen, brauchen im Allgemeinen länger als diejenigen, welche weniger effiziente Indices empfehlen. Unter den skalierbaren Ansätzen schneidet der kostenbasierte Vereinigungsalgorithmus am Besten ab (nur 1.5% schlechter als der naive (und gleichzeitig optimale) Ansatz). Dem kostenbasierte Vereinigungsalgorithmus folgt der ähnlichkeitsbasierte Vereinigungsalgorithmus und dann dann der Auswahlalgorithmus. Hinsichtlich der Ausführungszeit zeigt der kostenbasierte Vereinigungsalgorithmus das schlechteste Ergebnis, gefolgt vom ähnlichkeitsbasierte Vereinigungsalgorithmus sowie dem Auswahlalgorithmus.

Selbstadaptierendes Informationsverwaltungssystem

Im vorherigen Abschnitt wurde eine Menge von Algorithmen zur Empfehlung von Indexstrukturen diskutiert, die dem Designer einer verteilten Anwendung dabei assistieren, eine effiziente Menge an Indices zu definieren, mithilfe derer mehrdimensionale Bereichsanfragen auf Basis einer DHT abgearbeitet werden. Diese Algorithmen arbeiten *offline*, das heißt, es wird angenommen, der Workload, welcher den Algorithmen als Eingabe bereitgestellt wird, wurde in einem bereits bestehenden DHT-basierten Peer-to-Peer-System erfasst. Ferner wurde angenommen, dass die empfohlene Menge an Indexstrukturen manuell vom Designer der verteilten Anwendung installiert wird. Damit eine solche Installation durchgeführt wird, müsste das System allerdings abgeschaltet und neu gestartet werden, was – zumindest für kleine P2P-Anwendungen – praktikabel erscheinen mag, jedoch im Allgemeinen nicht erwünscht ist. Deswegen stellen wir im Folgenden Erweiterungen von OID zur Realisierung eines erweiterten und selbstadaptiven Peer-to-Peer-System vor. Das adaptive OID-System führt die Aufgaben der Empfehlung einer Indexstruktur sowie deren Installation *online*, sprich im laufenden Betrieb, durch und eliminiert damit die Notwendigkeit, die Menge an Indices manuell zu aktualisieren.

Am OID-System werden hierfür zwei grundlegende architektonische Änderungen vorgenommen. Die erste Änderung betrifft das Hinzufügen einer weiteren Komponente, welche als Anpassungs-Controller bezeichnet wird. Dieser Anpassungs-Controller ist ein Teil der

OID-Rahmenwerksschicht, der die oben diskutierten Algorithmen zur Empfehlung von Indexstrukturen ausführt. Die zweite Änderung betrifft die DHT-Schicht und erweitert die DHT einerseits um eine Funktion zum Senden von Rundsendenachrichten (engl. Broadcast) und andererseits um eine Funktion zur Aggregation von Werten mittels eines Broadcast-Aggregations-Baums [EAABH03, HZ10].

Das adaptive OID-System benutzt einen vierschrittigen Prozess zur Adaption der Indexstrukturen, welcher periodisch im System ausgeführt wird, um eine Menge von Indexstrukturen anzupassen. Für die Ausführung dieser Schritte werden drei Peer-Rollentypen definiert:

- **Adaptions-Peer** – Der Adaptions-Peer führt periodisch den Prozess zur Adaption der Indexstrukturen aus. Es darf nur einen Adaptions-Peer im Netz geben, der im Fehlerfall von der DHT ersetzt wird.
- **Monitoring-Peer** – Jeder Peer unseres Systems ist ein Monitoring-Peer. Die Menge des Monitoring-Peers ist zuständig für das lokale Sammeln von Suchanfragen des Workloads. Das heißt, jeder Monitoring-Peer protokolliert alle Anfragen, die er bearbeitet und stellt diese auf Anfrage zur Verfügung.
- **Sampling-Peer** – Sampling-Peers sind für das effziente verteilte Einsammeln des von Monitoring-Peers gesammelten Workloads verantwortlich. Jeder Peer kann die Rolle eines Sampling-Peers einnehmen.

Es folgen die vier Schritte im Prozess zur Adaption der Indexstrukturen:

1. **Verteiltes Einsammeln des Workloads** – Der Adaptions-Peer initiiert den Prozess zum Einsammeln des Workloads der mehrdimensionalen Bereichsanfragen von den Peers im Netz. Die Idee ist, beim Einsammeln eine ausreichend große Untermenge der Peers in unterschiedlichen Teilen des Netzes zu erreichen, um eine Annäherung und repräsentative Teilmenge der Gesamtmenge der Suchanfragen zu erhalten. Im ersten Schritt werden vom Adaptions-Peer β zufällige Peers im Netz kontaktiert. Die Peers werden daraufhin zu Sampling-Peers. Im zweiten Schritt kontaktiert jeder Sampling-Peer zufällig γ Monitoring-Peers im Netz und sammelt deren aufgezeichneten Workload von Suchanfragen ein. Diese Workloads werden anschließend an den Adaptions-Peer gesendet. Duplikate von Suchanfragen werden an den Sampling-Peers und an dem Adaptations-Peer mittels der global eindeutigen Bezeichner (IDs) der Suchanfragen herausgefiltert.
2. **Empfehlung einer Indexstruktur** – Sobald der Workload der mehrdimensionalen Bereichsanfragen vom Adaptions-Peer empfangen wurde, wird dieser an einen der Algorithmen zur Empfehlung einer Indexstrutkur, wie sie im vorherigen Kapitel diskutiert wurden, weitergereicht. Dieser Algorithmus empfiehlt daraufhin eine Menge an Indexstrukturen, die für den übergebenen Workload effizient ist.
3. **Adaptionsentscheidung** – Die dritte Phase, die vom Adaption-Peer ausgeführt wird, ist die Adaptionsentscheidungsphase. Das Ziel dieser Phase ist es zu bestim-

men, ob die Installation der empfohlenen Indexstrukturmenge nutzbringend ist oder nicht. Hierfür werden die geschätzten Kosten der aktuellen Indexstrukturmenge für den Workload zusammen mit den geschätzten Kosten der empfohlenen Indexstrukturmenge für den Workload verglichen. Die Kosten für die Installation der empfohlenen Indexstrukturmenge werden ebenfalls berücksichtigt. Der Adaptions-Peer initiiert die Installationsphase der ausgewählten Indices, falls die Kosten der aktuelle Indexsturkturmenge für den eingesammelten Workload größer sind als die Kosten der empfohlene Indexstrukturmenge für denselben Workload plus die Kosten für die Installation der empfohlenen Indexstrukturmenge. Die Installationskosten beinhalten die Kosten für das Rundsenden der empfohlenen Indexsturkturmenge und die Kosten für das erneute Indexieren der Datenobjekte.

4. **Installation der Indexstrukturen** – Damit das OID-System während der Installation der neuen Indexstrukturmenge weiterhin funktionstüchtig bleibt, wird die Indexstrukturinstallationsphase in drei Schritten vollzogen. In der ersten Phase wird die empfohlene Indexstrukturmenge an alle Peers gesendet, wobei die Rundsende-Funktion der DHT verwendet wird. Jeder Peer, der eine solche Nachricht erhält, beginnt damit, seine Daten erneut mithilfe der neuen Indexstrukturmenge zu indexieren. Sobald das erneute Indexieren beendet ist, sendet der Peer eine Bestätigungsnachricht an seinen Elternknoten im Broadcast-Aggregationsbaum. Im zweiten Schritt werden die Bestätigungsnachrichten aller Peers solange aggregiert, bis der Adaptions-Peer die letzte aggregierte Bestätigungsnachricht empfangen hat. Während dieser beiden Schritte werden die Suchanfragen weiterhin abgearbeitet, wobei die alte Menge von Indexstrukturen verwendet wird. Im letzten Schritt versendet der Adaptions-Peer eine Rundsendenachricht `Benutze Indexstrutkur`. Beim Empfang dieser Nachricht verwirft jeder Peer die alte Indexstrukturmenge sowie die zugehörigen Daten und beginnt fortan die neue Indexstrukturmenge für die Abarbeitung der Suchanfragen zu benutzen.

Die durchgeführten Evaluierungen zeigen, dass das adaptive OID-System fortlaufend die Indexstrukturmenge im System bezüglich des dynamischen Workloads von mehrdimensionalen Bereichsanfragen anpasst. Diese Adaptionen sind am nützlichsten, wenn es eine Vielfalt an unterschiedlichen Suchanfragen im System gibt. In den durchgeführten Evaluierungen zeigte das adaptive OID-System eine um mehrere Größenordnungen bessere Leistung im Vergleich zum nicht adaptiven System.

Anwendung: Ortsbezogene Informationsverwaltung

Abschließend stellen wir ein Peer-to-Peer-basiertes Informationsverwaltungssystem basierend auf den oben eingeführten OID-Konzepten vor. Als typische mehrdimensionale Bereichsanfrage unterstützt dieses System ortsbezogene Bereichsanfragen. Die wesentlichen Innovationen dieses Systems umfassen: (1) Die Benutzung einer Oktaeder-

Projektion mit geringen geometrischen Verzerrungen zur Repräsentation der räumlichen Daten, (2) die Verwendung der Sierṕinski SFC zur lokalitätsbewahrenden Zuordnung der Daten zu Peers, (3) eine Strategie zur Datenplatzierung, welche die Wahrscheinlichkeit einer Datenkonzentration an einzelnen Punkten des Netzes senkt.

Da das hier präsentierte räumliche Informationsverwaltungssystem auf dem oben eingeführten OID-System basiert, ist die Architektur des Systems die gleiche wie in Abbildung 1 dargestellt.

Die räumlichen Datenobjekte in unseren System sind definiert als Punkte auf der Erdoberfläche. Jedes Datenobjekt besteht aus zwei Primärattributen, `Latitude` und `Longitude`. Neben diesen Primärattributen kann jedes Datenobjekt weitere Attribute enthalten, die seine Eigenschaften beschreiben. Eine räumliche Bereichsanfrage in unserem System ist definiert als ein Polygon auf der Erdoberfläche. Das Anfragepolygon kann als Folge von Tupeln der Form $(\phi = value, \lambda = value)$ dargestellt werden, bei der ϕ und λ jeweils `Latitude` und `Longitude` repräsentieren.

Wie zuvor diskutiert ist die Lokalitätsbewahrung der räumlichen Daten die wünschenswerte Eigenschaft bei der Abbildung von Datenobjekten auf Peers. Wir erreichen diese Lokalitätsbewahrung durch eine Indexierung der räumlichen Daten auf Basis der Sierṕinski SFC. Da SFCs ein kartesisches Koordinatensystem benötigen, muss die Erdoberfläche von geografischen Koordinaten auf kartesische Koordinaten abgebildet werden. Wir verwenden zu diesem Zweck eine Oktaeder-Kartenprojektion (engl. Octahedral Projection) [Fur97], im Gegensatz zu einer Viereck-Projektion ((engl. Quadrilateral Projection)).

Die Oktaeder-Kartenprojektion erzielt eine gute Symmetrie hinsichtlich der Pole und die wichtigen globalen Merkmale (Pole, Äquator, Meridiane) werden von der Prjektion auf Seiten und Ecken von Dreiecken abgebildet [LT92]. Obwohl viereckige Karten einfacher zu verwenden sind, weisen sie stärkere Verzerrungen auf, sobald man sich vom Äquator in Richtung der Pole entfernt [IG01]. Eine Oktaeder-Kartenprojektion verzerrt dagegen weniger stark.

Jede Seite der Oktaeder-Karte ist ein Dreieck. Die Konstruktion der Sierṕinski SFC auf einem dreieckigen Teil der Oktaeder-Karte besteht aus zwei Schritten. Im ersten Schritt wird das Dreieck in kleinere Dreiecke zerlegt. Diese Zerlegung kann als rekursiver Prozess betrachtet werden, bei dem jeder Durchlauf des Prozesses das ursprüngliche Dreieck in zwei kleinere, gleiche Unterdreiecke teilt. Dieser Prozess wird k Mal durchgeführt. Im nächsten Schritt wird eine Linie gezeichnet, die den dreieckigen Ausschnitt der Karte betritt und wieder verlässt und dabei durch jedes Unterdreieck verläuft. Die Linie gibt eine Ordnung der Dreiecke vor, in der Reihenfolge ihres Durchlaufs. Der Datenplatzierungs-Controller benutzt diesen SFC-basierten Index, um die räumlichen Daten der Oktaeder-Karte mit Hilfe einer DHT auf Peers zu platzieren. Dieser Vorgang läuft wie in Abschnitt *'Architektur des OID-Systems'* beschrieben ab.

Falls eine einzelne Sierṕinski SFC benutzt wird, um alle räumlichen Datenobjekte auf der Oberfläche der Oktaeder-Karte zu indexieren, könnte dies zu einer ungleichförmigen

Verteilung der Daten über die DHT und somit die Peers führen. Um den Grad der ungleichmäßigen Verteilung der Daten zu reduzieren, nutzen wir eine eigene SFC für jede Seite der Oktaeder-Karte. Allerdings bringt die Verwendung einer eigenen SFC für jede Seite auch einen Nachteil: Die Lokalität der Daten entlang der Kante an der zwei Seiten aufeinander treffen wird dadurch gestört. Bei der Verwendung einer einzigen SFC würden zwei räumliche Datenobjekte, die entlang einer Kante liegen, an der zwei Seiten zusammentreffen, Bezeichner erhalten, die numerisch nahe beieinander liegen.

Die Anfrageverarbeitung verfeinert die räumliche Bereichsanfrage mit Hilfe der Sierṕinski SFC, um die Bezeichner zu ermitteln, die durch die DHT angefragt werden sollen. Die Peers, die für diese Bezeichner zuständig sind, werden wie bereits im vorigen Abschnitt beschrieben mittels eines baumbasierten Vermittlungsansatzes kontaktiert, der die Anzahl paralleler Anfragenachrichten im Netz reduziert.

Das ortsbezogene Informationsverwaltungssystem wurde mit zwei unterschiedliche Arten von Systemkonfigurationen evaluiert. Die erste Konfiguration benutzt eine eigene Sierṕinski-SFC für jede Seite der Oktaeder-Karte. Die zweite benutzt eine einzelne durchgängige Sierṕinski-SFC für die gesamte Karte. Unsere Evaluationen zeigen, dass im Vergleich die erste Konfiguration eine bessere Verteilung der Daten über die Peers erzielt. Allerdings ist Leistung bei der Anfrageauflösung in der zweiten Konfiguration besser. Unsere Evaluierungen zeigen außerdem, dass der optimierte Anfrageauflösungsalgorithmus eine Verringerung der parallelen Nachrichten um bis zu 96% erreicht, bei einer Anfrage an eine Fläche der Größe Deutschlands mit einem Netz von 100.000 Peers.

Chapter 1

Introduction

1.1 Motivation

During the past decade, peer-to-peer (P2P)-based information exchange has emerged as a new communication paradigm. Unlike the traditional server/client architecture that includes a server offering services to clients, P2P architecture is based on the notion of equality. Each device (peer) in a P2P system is involved in distributed implementation of services such as data storage, information retrieval, distributed computing etc.

P2P systems have appealing characteristics such as high availability, anonymity, fault-tolerance, self-organization, and scalability. A P2P infrastructure could be used to store and share information efficiently. Vast user resources could be utilized in such a system. P2P systems are cheap to set-up and they enable everyone to share information whether it is as simple as contact information or as complex as location coordinates.

Information discovery in P2P systems takes place by propagating user queries in an overlay network that is constructed on top of the IP network. We classify the queries in the P2P overlay networks into two major classes (see Figure 1.1): simple queries and complex queries. Simple queries are exact match queries, while complex queries could be further classified into following major sub-classes: range queries, multi-attribute queries, similarity queries and aggregation queries. A complex query could fall under multiple sub-classes. Following are some examples of user queries in a P2P overlay network:

- "*Retrieve contact information of all people of age* 25". This is an exact match query because, an exact value for the *age* attribute is specified.
- "*Find all music albums released between year* 1999 *and* 2001". This is a range query because, a range over the *year* attribute is specified.
- "*Find all restaurants in the city of Stuttgart that serve Sushi*". This is an example of a multi-attribute query comprised of attributes *city* and *food type.*

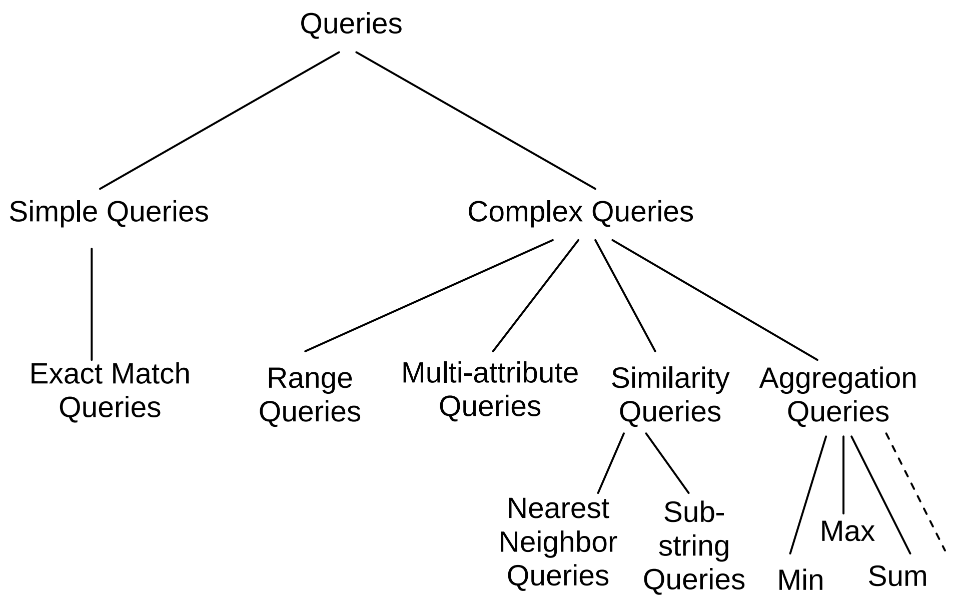


Figure 1.1: Query Classification - The dashed line denotes that there are more type of aggregation queries

- "*Find all computers with CPU speed between* 1.0 *and* 4.0 *GHz & RAM between* 2 *and* 6 *GB*". This is a multi-attribute range query since a range of values is specified for at least one attribute (e.g. *RAM*) and there are more than one attributes in total.

User queries in P2P overlay networks have been generally supported by two main types of network topologies: unstructured overlay networks and structured overlay networks. Unstructured P2P overlay networks impose no restrictions on the overlay network topology. New peers join the network by connecting to one or more existing peers in the network. These type of P2P overlay networks provide anonymity to the user and have proven to be a good choice for sharing popular information. Unstructured P2P overlay networks support simple as well as complex queries. However, information discovery in these networks is based on best-effort search i.e. there are no guarantees that the searched information would be found if it exists in the network. Typical examples of unstructured P2P overlay networks are Gnutella [CCR05], FreeHaven [fre10], and Kazaa [kaz10].

Structured P2P overlay networks impose strong restrictions on the overlay network topology. These type of overlay networks guarantee that a query would be routed to any peer in the system in a pre-defined maximum number of overlay hops. Structured P2P overlay networks provide efficient support for simple queries using Distributed Hash Tables (DHT)s. However efficient support for more complex queries such as multi-attribute range queries is an open research question.

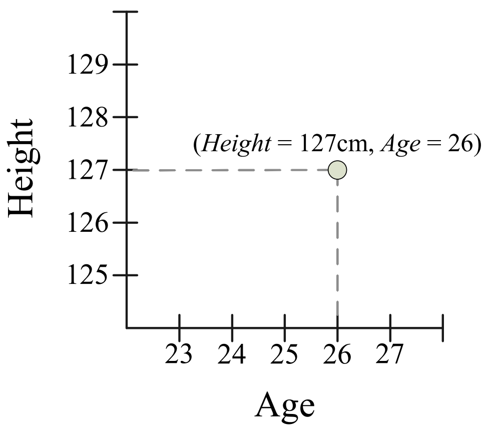


Figure 1.2: An example of a 2-dimensional attribute space

1.2 Problem Statement and Contribution

Modern P2P applications such as resource discovery in grid computing [MRPM08], P2P video streaming [ND08], and spatial information discovery [MTD+09], require support for multi-attribute range queries. Since unstructured P2P overlay networks fail to provide guaranteed information discovery, despite the fact that they support a rich set of queries, structured P2P overlay networks, specifically DHTs, have been extended to support multi-attribute range queries.

The support for multi-attribute range queries in DHTs has been realized by installing data indices over them. These are the same data indices that have been traditionally used by Database Management Systems (DBMS) for storing and searching the data efficiently in a relational database [GMUW08]. The basic functionality of a data index is to organize the data in a manner that, storing and searching the data does not require traversal of the complete database.

Structured information discovery systems such as [CRR+05, GYGM04, SP03] make use of Space Filling Curves (SFC)s [ARR+95] as data indices to support multi-attribute range queries over DHTs. A SFC is a dimension reducing function that linearises multi-dimensional data to a single dimension. Consider a data object as a list of attribute/value pairs, e.g., "*Height* = 127cm, *Age* = 26, . . . ". If each attribute in such a data object is considered as a dimension and the attribute value is considered as a coordinate, then each data object can be viewed as a point in a multi-dimensional space (see Figure 1.2). A SFC is a line that passes through this space linearising each point in it, while preserving data locality, i.e. points close to each other in the space are mapped close on the SFC line with a high probability.

The dimension reducing property of SFCs allows mapping of multi-dimensional data objects to a single dimension. The 1-dimensional data objects can then be placed in a network of peers using a 1-dimensional DHT such as Chord [SMK+01]. Furthermore, the locality preserving property of SFCs maps data objects that are close in a multi-dimensional space to a close neighbourhood of peers in a DHT. This enables efficient processing of range queries because the queries could be resolved by a set neighbouring peers in the network.

1.2.1 Problem Statement

Generally, there have been two types of approaches for indexing multi-dimensional data using SFCs in DHTs. The first approach indexes the combination of all data attributes [CRR+05, GYGM04, SP03]. Information discovery systems that use this approach require that the multi-attribute range queries contain ranges over all attributes, which is not a realistic assumption for all P2P applications. Attributes that do not appear in queries are considered to be wild-cards, and the performance of these systems deteriorates with increasing number of wild-cards in queries. The second approach indexes each data attribute individually [AX02, CFCS03, SOTZ05, TP03]. Using this approach, a multi-attribute range query is resolved in one of the following two ways: either the query is performed on a single attribute and then the data is filtered at the contacted peers, or the query is divided into multiple single-attribute range queries and each query is performed individually; later, the data is filtered at the query initiator. In the first case, the query incurs high latency due to filtering at each peer. Moreover, a large number of peers are contacted that may or may not have the data objects that match the queried ranges for all attributes. In the second case, a large number of data objects are transferred to the query initiator, only some of which match the queried ranges for all attributes.

Due to the shortcomings of the systems discussed above, a P2P information discovery system that uses SFCs for supporting multi-attribute range queries in an efficient and scalable manner is needed.

1.2.2 Contribution

We present the Optimized Information Discovery (OID) system that enables multi-attribute range queries over DHTs in an efficient and scalable manner. The OID system indexes the data objects in a P2P overlay network using several SFC-based indices with each index using a certain attribute combination. A multi-attribute range query is resolved in two steps. First, the query is mapped on each SFC-based index to determine the index with the least network cost. Next, the search for matching data objects is initiated in the network using the least expensive index.

In order to realize the OID system, following main contributions have been made by our research:

- The basic OID system has been developed that optimizes the overall system performance for popular multi-attribute range queries issued by the P2P applications [MTD+08].
- Two types of distributed query optimization algorithms have been developed for scalable and efficient processing of multi-attribute range queries in DHTs [MTD+08].
- Several index recommendation algorithms have been developed that help the designer of a P2P application, in defining optimal SFC-based indices [MDR10].
- An index adaptation process has been designed to automatically update the installed set of indices according to the changing set of multi-attribute range queries in the system [MTDR10].
- An extension of the OID system, for supporting spatial range queries in DHTs, has been developed [MTD+09].

1.3 Structure

The rest of the document is structured as follows:

- In Chapter 2, we present a detailed background of information discovery in P2P overlay networks. Moreover, we discuss the concepts of distributed hash table and space-filling curve in detail.
- In Chapter 3, we introduce the basic architecture of the OID system with the focus on optimizing popular multi-attribute range queries issued by P2P applications. The distributed query optimization algorithms for efficient and scalable query resolution are also discussed in this chapter.
- In order to optimize the overall system performance for multi-attribute range queries, three index recommendation algorithms are presented in Chapter 4. These algorithms help the designer of a P2P application in defining optimal indices for the application, without considering any specific query popularity distribution.
- An index adaptation process, for continuously optimizing the overall system performance for multi-attribute range queries, is discussed in Chapter 5. This adaptation process automatically optimizes the performance of the OID system according to the dynamic set of multi-attribute range queries in the system.
- In Chapter 6, we present an spatial information discovery system that is based on the OID system for performing spatial range queries using geographic coordinates.
- Finally, the summary of our work and an outlook on possible future work are presented in Chapter 7.

Chapter 2

Background

In this chapter, we present the concepts that form the basis of our research. We start by introducing the peer-to-peer (P2P) computing paradigm (Section 2.1), followed by a detailed discussion on unstructured (Section 2.2.1) and structured (Section 2.2.2) P2P information discovery systems. Finally, we present the concept of Space-Filling Curve (SFC) and discuss the Hilbert SFC in detail (Section 2.3).

2.1 Peer-to-Peer Overlay Networks

The era of networked computing began with the client/server architecture. Originally, the client/server architecture consisted of a single server providing services to multiple clients (see Figure 2.1 (a)). A typical example of such a service is a network file system (NFS). NFS allows clients to connect to a network storage device where they could store and access data remotely.

With advancements in computer networks, the historical client/server architecture was leveraged using distributed computing. Distributed computing allowed a set of servers to collaborate in order to provide services to multiple clients. This made the system more robust and fault tolerant. Typically, the servers in the client/server architecture used to be resource-rich computers with a high network bandwidth. Clients on the other hand, were resource-scarce computers with low bandwidth. However, this situation changed with technological advances in computing hardware and high-speed internet being available to the general public. The clients became increasingly resource-rich computers which led to the era of peer-to-peer (P2P) computing and the P2P systems. A P2P system could be defined as:

"*A peer-to-peer system is a self-organizing system of equal, autonomous entities(peers) which aims for the shared usage of distributed resources in a networked environment avoiding centralized servers.*" [SW05].

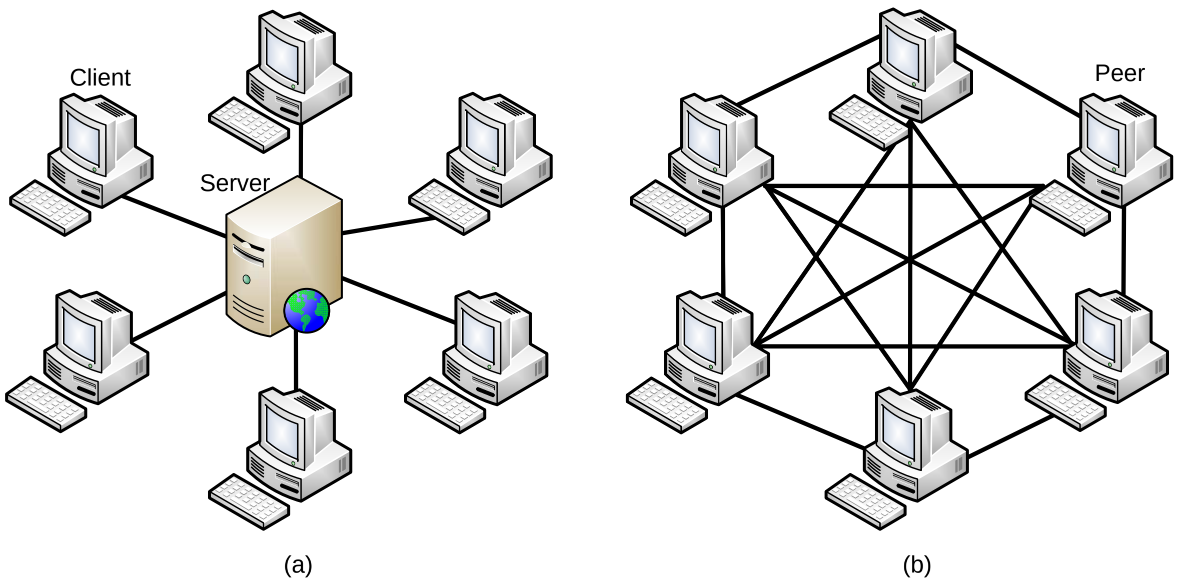


Figure 2.1: (a) Client/Server Architecture, (b) Peer-to-Peer Architecture

The P2P architecture consists of several computers (peers) connected to each other to form an overlay network (see Figure 2.1 (b)). A peer not only uses a service but also participates in providing that service to the other peers, i.e., a peer acts as a client and as a server. A typical example of a P2P service is a file-sharing service, such as Gnutella [gnu10]. Using Gnutella, a peer can share its local files with other peers, thus employing the role of a server. Moreover, a peer can download files shared by other peers acting as a client.

The P2P architecture has several advantages over the traditional client/server architecture. P2P overlay networks are cheap to set up as there is no need for expensive high-end machines, i.e., dedicated servers. Instead, P2P overlay networks rely on resource aggregation of several cheap machines with moderate hardware capabilities to provide the server functionality. Due to the lack of servers, the need for a system administrator is also eliminated. Each peer is administered by its own user. Moreover, since there is no central entity, no single machine or a small set of machines (servers) can collect information about all the peers in the network which allows peers to stay relatively anonymous.

Apart from the advantages mentioned above, P2P overlay networks also have some disadvantages. Security is one of the major concerns in P2P overlay networks. Due to the lack of central administration, malicious peers can join the network and disrupt the whole system. Reachability of a high-end peer could be reduced by a set of surrounding low-end peers. Moreover, simultaneously joining and leaving peers could cause the network to become partitioned. Furthermore, due to versatility in the capabilities of peers, no performance guarantees could be provided in a P2P overlay network.

2.2 Peer-to-Peer Information Discovery Systems

P2P overlay networks have been extensively used for information storage and discovery purposes. The first step towards P2P-based information exchange was taken by Napster [SGG03] that implemented a file-sharing service. The Napster network consists of a server that acts as a directory service. Each peer in the network connects to the server and sends the list of its local files to the server. Each list and the IP address of the corresponding peer is stored in a database on the server. Any peer in the network can search for a file by sending a query to the server. The server performs a local database lookup using the received query and responds with the IP address of the peer hosting the queried file. The query initiator then directly connects to the peer hosting the file using TCP and initiates the file transfer.

Due to the use of a central server as a directory service, Napster is not a pure P2P system in a strict sense. If the central server crashes, the whole network is compromised. However, the data exchange in Napster falls under the P2P communication paradigm. Therefore, Napster has been termed as a hybrid P2P system. Pure P2P systems of the unstructured and the structured forms are discussed in the following sections.

2.2.1 Unstructured Peer-to-Peer Systems

The popularity and success of Napster fuelled the research and development in the area of P2P-based information exchange. Soon after Napster, the first pure P2P information discovery system Gnutella 0.4 [Rip01] was released. Gnutella 0.4 is an unstructured P2P system. In an unstructured P2P system, the topology of the network is decoupled from the content of the peers, i.e., the network does not control the content placement. The peers organize themselves to form a small-world network topology [MN04]. In a small-world network, there exists a shortest path from each peer to any other peer in the network. Therefore, the aim of the unstructured P2P systems is to utilize these paths efficiently for searching the content in the overlay network.

There are two main categories of content search algorithms in unstructured P2P systems, namely, blind search algorithms and informed search algorithms. When using a blind search algorithm, peers have no knowledge about the location of the searched objects in the network. The search is solely based on local connectivity information, i.e., the neighbouring peers. However, informed search algorithms rely on the information collected about the location of the objects in the overlay network. These algorithms are heuristics that based on the object location information, choose a path in the network that maximizes the probability of finding the searched data.

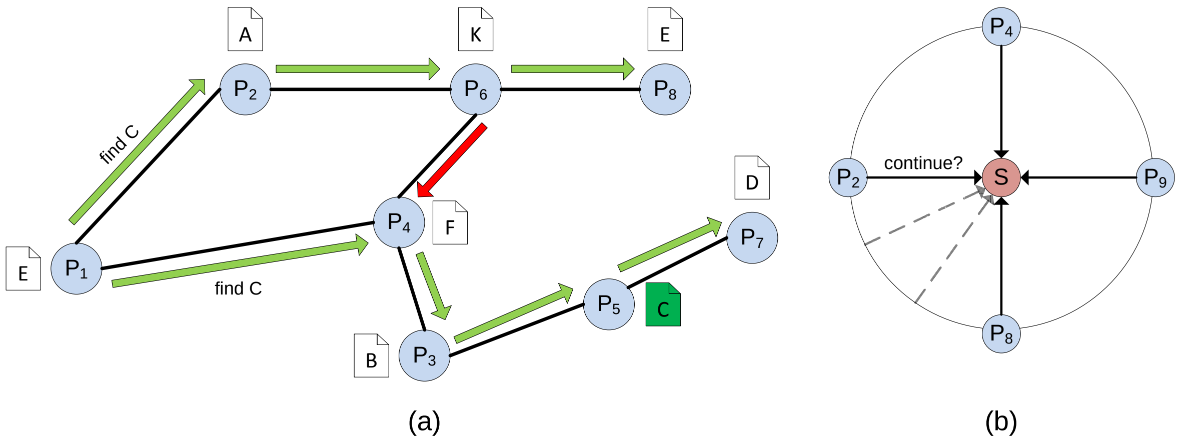


Figure 2.2: (a) Breadth-first Search, (b) Network Implosion Problem

2.2.1.1 Uninformed Search

The content search in Gnutella 0.4 was introduced as a modified version of the breadth-first search (BFS). In a BFS scenario, a peer forwards an incoming query to all its neighbours except the peer that it received the query from. A query that has been previously seen by a peer is dropped as well. This prevents the scenario where a query circles the network endlessly. Figure 2.2(a) shows routing of a query using BFS in an unstructured P2P system. Only a single query message is send from peer P_4 to peer P_3 because the duplication query message is dropped at P_4.

BFS has a high success rate, i.e., it finds all the matching objects in the network in cases where there is no network failure. However, BFS may lead to complete network being flooded by a query even if the matching objects are found in few steps because there is no mechanism to stop the search at an early stage. If each peer asked the query initiator whether to continue the search then the network will face an implosion problem (see Figure 2.2(b)). Moreover, the number of searched peers grows often exponentially and peers may receive unnecessary redundant copies of the same query message from different neighbours.

To avoid the short-comings of the general BFS algorithm, Gnutella 0.4 modified the BFS algorithm and introduced a Time to Live (TTL) with each query initiated at a peer. This modified version of the BFS algorithm is known as limited flooding [TR06]. Each peer that receives a query first decrements the TTL value associated with the query and then forwards the query to its neighbours. If the value of TTL reaches 0, the query is dropped. Although introduction of TTL limits the flooding of the network, it leads to a query horizon (see Figure 2.3(a)). Due to the presence of a query horizon, a query reaches only certain parts of the network when initiated at a particular peer. For example, in Figure 2.3(a) a query initiated at peer S only reaches until the peer P_9 with the maximum TTL. Therefore, peers P_2, P_4, and P_8 lie outside the range of S.

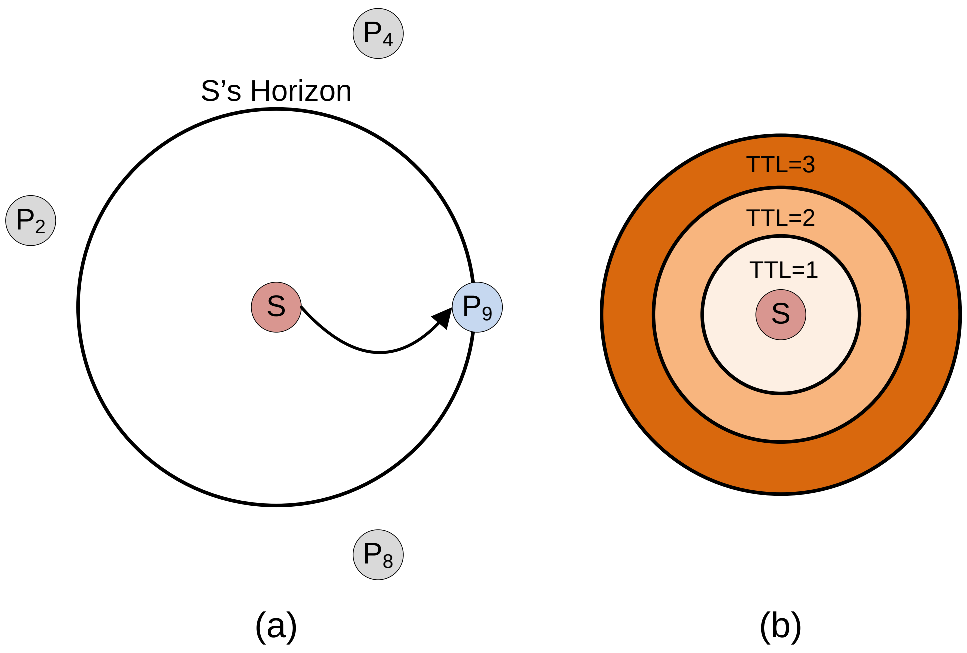


Figure 2.3: (a) Limited Flooding, (b) Iterative Deepening

Choosing an appropriate TTL value is not a trivial task as well because a small TTL value leads to small network coverage, while a large TTL value induces a high network load due to flooding.

A variation of limited flooding known as iterative deepening has been presented in [YGM02]. Using iterative deepening a peer specifies multiple TTL values for propagating a query through the network. Each TTL value is larger than the former one. A query is initiated by a peer using the BFS with the minimum TTL. Once the TTL reaches 0, the query gets frozen at the query horizon. If the query has been satisfied, the frozen query gets dropped after a timeout. Otherwise, the query initiator propagates a *Resend* message through the network until it reaches the peers at the query horizon. These peers then unfreeze the query by adjusting the TTL to the next specified value. This process continues as long as a query is not satisfied or until the maximum TTL value is reached. Figure 2.3(b) shows the expanding query horizon in an iterative deepening scenario with TTL values from 1 to 3.

Iterative deepening solves the problem of choosing an appropriate TTL. Popular objects could be found very quickly with small TTL values, while unpopular objects may be found if the maximum TTL value is sufficiently large. The main disadvantage of iterative deepening is that it increases the delay of finding the objects because there are x number of searches instead of just one, where x is usually larger than one. Moreover, the load induced by iterative deepening is still high because the number of peers visited still increases exponentially. Also, the problem of peers receiving redundant query messages is not resolved.

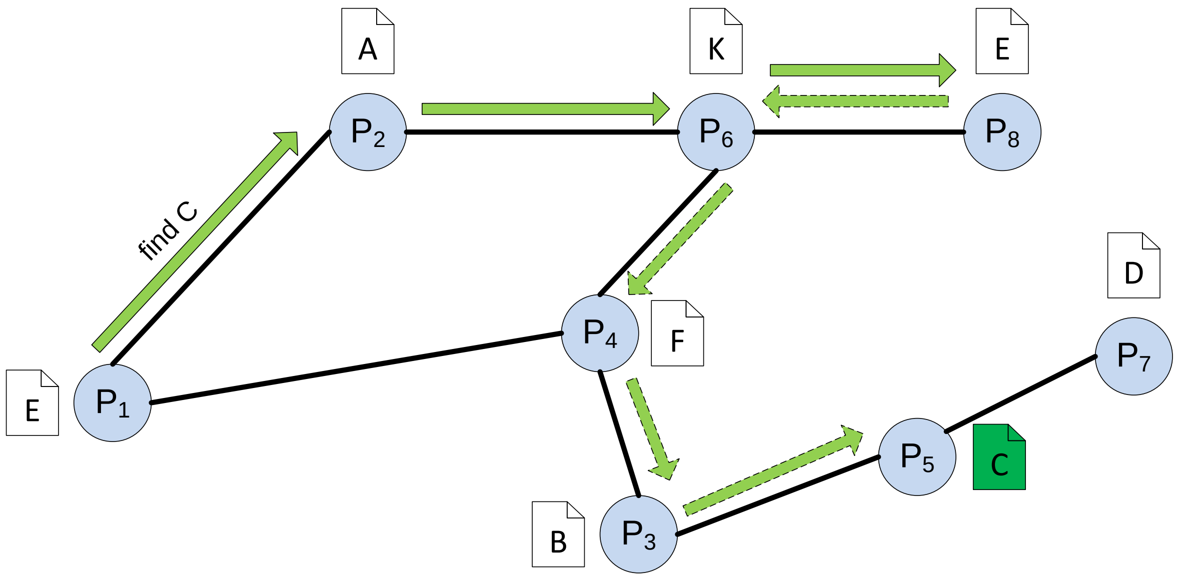


Figure 2.4: Depth-first Search

Another variation of BFS algorithm is the randomized BFS algorithm [KGZY02]. Using the randomized BFS algorithm a peer forwards a query to a subset of its neighbours instead of sending it to all of its neighbours. In order to choose this subset, a peer first generates a random number p_i between 0 and 1 for each of its neighbours. If the generated number is less than a specified application-level threshold $P_{threshold}$, the query is forward to the neighbour, otherwise it is not. $P_{threshold}$ could be chosen such that statistically duplicate queries are eliminated and the connectivity is preserved.

Compared to the previously discussed BFS algorithms, the randomized BFS algorithm significantly reduces the network load because it avoids redundant queries. However, the absolute network load is still high because the number of peers visited still increases exponentially. Moreover, choosing an appropriate value for $P_{threshold}$ is not straightforward because a small value results in a low query success rate, while a large value induces a high network load due to flooding.

Another form of uninformed search in unstructured P2P systems is the depth-first search (DFS) [D1̈0]. In a DFS scenario a peer forwards a query to all its neighbours sequentially, i.e., there exists only a single query message per query in the network. When the query reaches a peer with no unvisited neighbours, it is back-tracked until a new unvisited path is found. The search ends when the query reaches the initiator and there are no unvisited paths in the network. Figure 2.4 shows propagation of a query using the DFS in an unstructured P2P system. For simplicity, the search has been shown to stop when the searched object is found. Note that the query starts to back-track when it hits a dead-end at peer P_8.

DFS has a high success rate because the query reaches all the peers in the system.

However, the latency of the search is also high because there is no parallel processing of the query like in the BFS. Moreover, due to back-tracking, the problem of peers receiving duplicate messages is also not resolved.

A popular variation of the DFS is the random walk [LCC+02]. In a random walk scenario, a peer forwards a query to one of its randomly chosen neighbours. A peer could initiate k random walks simultaneously. One query message in this case would be a single random walker. Initiating k random walkers reduces the latency of the search because k random walkers reach approximately the same number of peers in i steps as a single random walker in ki steps. This reduces the query latency by a factor of $1/k$ however, it increases the network load by a factor of k.

A random walk ends when a termination condition is met. A termination condition could either be TTL-based or check-based. In a TTL-based termination condition, each random walker is assigned a TTL and the walker terminates when the TTL reaches zero. In a check-based termination, a walker periodically checks with the query initiator if the query has been satisfied before going to the next hop. If a query has been satisfied, the walker terminates otherwise, it continues. Note that, checking with the query initiator does not lead to network implosion if the number of random walkers is small.

Compared to the standard flooding scheme, random walk reduces the network load at least by an order of a magnitude [LCC+02, TR03]. Moreover, due to the random selection of the neighbours, the load of query processing is fairly balanced among the neighbours of a peer. The major disadvantage of the random walk is its variable performance because the success rate depends on the structure of the network as well as the random choices made during the walk.

In order to increase the efficiency of random walks and to reduce the number of redundant query messages received by a peer, a simple optimization that maintains the state of the random walks could be implemented. Using this optimization, each peer maintains the query ID and the ID of the neighbour which the query was previously forwarded to. If the same query arrives at the same peer again, it is forwarded to a different neighbour.

Most of the uninformed search techniques discussed above could be combined in several ways to form hybrid blind search strategies. For example, a combination of random walk and limited flooding could be developed such that, each peer that is visited by a random walker does a shallow flooding with TTL = 1. The advantage of this approach is that the search latency of traditional random walk is reduced at the expense of higher network load because high-degree peers can search many neighbours in a single step.

2.2.1.2 Informed Search

In order to improve the efficiency of content search in unstructured P2P systems, several informed search algorithms have been proposed. The distinguishing characteristic of the informed search algorithms is that they utilize meta-information, maintained at each peer, to direct the queries towards paths that would yield best results. The meta-information could be based on: the number of previously returned results, degree of the neighbouring peers, type of queries answered by neighbouring peers, etc.

Using directed BFS [YGM02], which is an informed version of the BFS algorithm, each peer maintains meta-information about the number of results previously returned by each neighbour. Alternatively, a peer could store the latency to all its neighbours. In order to perform content search, the query initiator sends a query only to those neighbours that could yield the best performance based on the number of previously returned results or latency. These neighbours in turn send the query to all of their neighbours just like in a BFS.

Directed BFS reduces the search latency because the queries are directed towards peers that are most likely to respond with positive results. Moreover, since queries are not sent in all directions by the query initiator, directed BFS induces less network load compared to the general BFS algorithm. However, the storage cost increases because each peer has to store some meta-information about all its neighbours. Another major disadvantage of directed BFS is that the search load is not uniformly distributed among the peers in the network. Peers that perform well have to do even more which could cause them to become overloaded and result in a network bottleneck.

Intelligent search mechanism [KGZY02], also known as the intelligent BFS, enables each peer to maintain a profile for each of its neighbours. The profiles are based on previously answered queries by the neighbours. Upon receiving a query, a peer compares it against the profiles of all its neighbours. The query is then forwarded to m number of peers that are most likely to respond with positive results. Intelligent search mechanism induces less network overhead compared to the directed BFS because each peer sends the query only to a subset of its neighbours. However, like the directed BFS, the storage cost increases due to profiling of the neighbours and the query load is also not uniformly distributed through the network.

Using adaptive probabilistic search (APS) [TR03], each peer maintains a probability value for routing a request for an object o to each of its neighbours. Search for an object is initiated by the query initiator in form of k independent random walkers. For each hop of a random walker, if a peer cannot answer the query, it forwards the query to a neighbour with the highest probability value. Once the requested object is located by a random walker or the walker terminates unsuccessfully, a response is sent through the reverse path so that the intermediate peers can update the probability values accordingly. If the search is successful, the relative probability for the searched object is increased at each peer on the reverse path. Otherwise, the probability is decreased.

Compared to the uninformed random walk, APS significantly improves the latency for finding an object. However, the network load is higher because a response is sent through reverse path to the query initiator. Nonetheless, APS achieves high success rate with low bandwidth consumption and is highly adaptive to the network topology changes. The major disadvantage of APS is high storage consumption due to profiling.

Adaptive resource-based probabilistic search (ARPS) [ZZSL07] uses weighted probabilities to route a query for an object o through the network. For each of the previously requested objects, a peer maintains a popularity estimate for the object. The popularity is based on the previously received responses. Upon receiving a query for an object, a peer uses its own degree and the popularity of the object to calculate an appropriate forwarding probability for the query. The higher the popularity of the searched object, the lower the forwarding probability that is calculated for the query, and vice versa. Additionally, if the forwarding node is a high degree node, the weighted forwarding probability is smaller compared to a low degree node. If the popularity of the searched object is unknown to the peer, the peer forwards the query to all its neighbours with a forwarding probability of 1, otherwise, the query is forwarded to some neighbours with an appropriate forwarding probability.

ARPS efficiently searches popular data objects in the network without inducing a high network load. However, for unpopular items, the network load could be much higher compared to APS, because most of the peers forward the query to all their neighbours, in effect, partially flooding the network. ARPS replaces random selection of neighbours with a probabilistic selection which is more efficient for searching popular items through the network. However, probabilistic selection also leads to non-uniform query load distribution.

Percolation search [SBR04] is a distributed informed search algorithm that allows processing of complex queries in unstructured P2P systems with a power-law graph topology. The search consists of following three major steps: (I) Content Cashing – A peer performs a short one-time-only random walk and replicates its content list on each of the visited peers, (II) Query Implantation – Search for content is prepared through query implantation when the query initiating peer performs a short random walk and implants the query at each of the visited peers, (III) Bond Percolation – The actual search for content begins when the peers with the query implant broadcast the query to each of their neighbours with a probability q, where q is based upon a percolation threshold and the exponent of the power-law graph distribution.

The developers of the percolation search show that any content in the network could be found with probability one in O(log N) time, where N is the network size. However, it is not clear, how the search reacts to the changes in the content list of peers since the content list replication is performed only once. Moreover, percolation search is suitable only for networks with an exponent of 2 or 3 of power-law distribution.

Hybrid periodic flooding (HPF) [ZLXN03] is a modified version of iterative deepening that eliminates the partial coverage problem while avoiding the unnecessary traffic associated with flooding at the same time. Each peer in HPF uses a periodical function

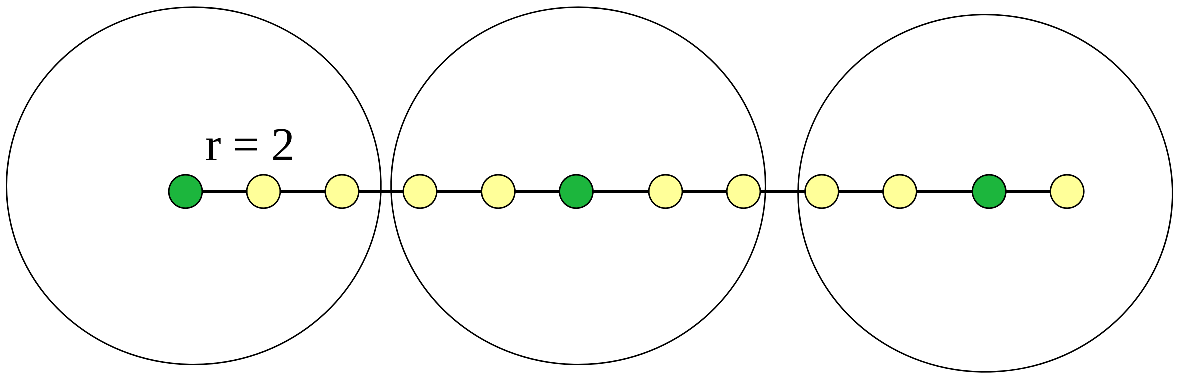


Figure 2.5: Local Indices

to determine the number of neighbours a message should be forwarded to. The periodical function could be based upon several factors such as bandwidth, communication cost, number of previously returned results, etc. The TTL-based search termination condition of HPF is the same as the termination condition for iterative deepening (see Section 2.2.1.1).

Due to the iterative nature of HPF, the partial coverage problem is significantly reduced. However, the search latency increases when compared to non-iterative search algorithms. Query load-balancing seems to be an issue with HPF as well.

The GIA protocol [CRB+03] mainly focuses on balancing the query load among peers so that the probability of query hot-spots in the network remains low. GIA uses biased random walk to resolve queries. In order to avoid query hot-spots, a peer sends a query to a neighbour only if the neighbour allows it to do so. The neighbour provides this permission in form of flow-control tokens. Each peer in a GIA network has a number of flow-control tokens associated with it. The flow-control tokens represent the capacity of the peer. When forwarding a query, a peer selects the neighbour with the highest number of flow-control tokens as the next hop of the random walk. If there are no flow-control tokens available, the peer queues the query locally until a flow-control token is received from a neighbour. GIA also implements following two search optimizations: (I) Bookkeeping – Each peer logs the information about each query that has been previously sent to each neighbour. Using this information, a query is not sent to the same neighbour again, (II) 1-hop Replication – The list of the data objects at a peer is replicated at each of its neighbours. Using this information a random walk could be terminated a hop earlier.

Unlike most of the informed search algorithms, GIA successfully achieves query load-balancing among peers and avoids query hot-spots in the network. However, query load-balancing comes at the cost increased network load due to exchange of flow-control tokens. Moreover, since the random walk is biased, the low capacity nodes might be eliminated from search making the unpopular items shared by them unsearchable.

A popular approach for content search in unstructured P2P systems is using local indices [YGM02]. Each peer in this approach keeps a local index over the data of all peers within r hops from itself, where r stands for radius. This way, a peer can answer a query on behalf of several peers in the neighbourhood reducing the query processing time, and therefore the query latency. A peer announces its content through flooding with a TTL equal to r. Search for content is performed using the BFS. However, in contrast to the BFS, the query is processed only at peers that are $2(r + 1)i$ hops away from the query initiator. In an example scenario shown in Figure 2.5, where $r = 2$, peers that process the query are marked using a darker colour.

Local indices provide the same query results as the BFS but with a reduced query latency because every peer in the network does not process the query. However, the reduced query latency comes at the cost of increased network load due to flooding of content announcements. Therefore, this approach becomes exceedingly unscalable with highly dynamic systems.

At the same time when local indices were introduced, a similar but more efficient approach for content search in unstructured P2P systems called routing indices (RIs) [CGM02] was also suggested. Using RIs each peer builds an index structure containing aggregated meta-information about the number and the type of documents available on each path through its neighbours. This information is later used to direct a query towards selected paths that would yield best results. Following three types of RIs were introduced: (I) Compound RIs (CRIs) – Using a compound RI, each peer maintains the aggregated number of documents available on each topic through each of its neighbours, (II) Hop-count RIs (HRIs) – Using hop-count RIs, a peer maintains the aggregated number of documents on each topic for each hop through its neighbours up to a maximum number of hops, (III) Exponential RIs (ERIs) – In order to reduce the size of HRIs, ERIs could be built by aggregating the information in HRIs at the cost of loss of accuracy.

All three RI approaches have some advantages and drawbacks. CRIs are smaller compared to HRIs but their major disadvantage is that the latency of query resolution cannot be estimated. HRIs have large storage overhead but allow a peer to estimate the latency of query resolution. ERIs are basically compact HRIs that lose the accuracy that HRIs provide.

There are several other informed search approaches that are similar to the ones described above and are therefore not discussed here. Some of these approaches are: activity based search [YLY07], preferential walk [ZCS05], equation based adaptive search [BA07], distributed resource location protocol [MK02], Gnutella with efficient search [ZH06], high degree random walk search [YS07], reinforcement learning based search [LW06], and distributed search technique [TS10].

2.2.2 Structured Peer-to-Peer Systems

Unstructured P2P systems in general, have been shown to be efficient for sharing popular content. However, there are some major drawbacks of these systems. For example, for most of the approaches discussed above, discovery of an data object cannot be guaranteed even if it exists in the network and there are no network faults. This problem gets even worse for unpopular data objects. Moreover, approaches such as flooding, BFS and DFS, that guarantee the discovery of all searchable objects, are either inefficient or unscalable.

In order to avoid these drawbacks, and to provide concrete guarantees regarding object discovery, structured P2P systems have been developed. Unlike unstructured P2P systems, content in structured P2P systems is precisely placed at specific peers in the network. The search for specified content is therefore directed towards the peers that store that content. Moreover, the network topology, i.e., the neighbourhood of a peers, is also tightly controlled in structured P2P systems.

A structured P2P system is typically implemented as a distributed hash table (DHT). In a DHT, each data object and each peer is assigned a unique identifier. A data object is typically represented as a key/value pair. The key of the data object is hashed to generate an identifier for the data object, e.g. Hash(key = “john”) = 80. The value of the data object is either the data itself (direct storage) or a pointer to the location of the data (indirect storage).

Each peer in a DHT is a bucket of hash values, i.e., each peer is responsible for certain range of hash values. For example, in a network of three peers, peer ‘A’ could be responsible for hash values in range (0, 100), peer ‘B’ could be responsible for hash values in range (101, 195), and peer ‘C’ could be responsible for hash values in range (196, 255). A peer stores the data objects that fall under its range. Therefore, a data object with Hash(key = “john”) = 80, would be stored at peer ‘A’.

Peers in a DHT have a limited view on the distributed index structure, i.e., a peer only knows the ranges of few other peers in the network. Moreover, each peer has some links to other peers in the DHT. Using these links, search for a data object is forwarded to a peer that is responsible for the hash value of the data object’s key.

DHTs make use of consistent hashing functions opposed to standard hashing functions. Since each peer in a DHT is a hash bucket, if standard hashing were employed, any changes in the total number of peers would require rehashing of all the data objects in the network. This would make DHTs unscalable because a certain churn rate is always associated with a P2P network. Consistent hashing avoids this problem and requires rehashing of only $O(K/N)$ data objects when a peers joins/leaves the DHT, where K is the total number of keys and N is the total number of peers in the DHT.

Several types of DHT routing geometries have been proposed in the past. Here, we discuss some of them in detail.

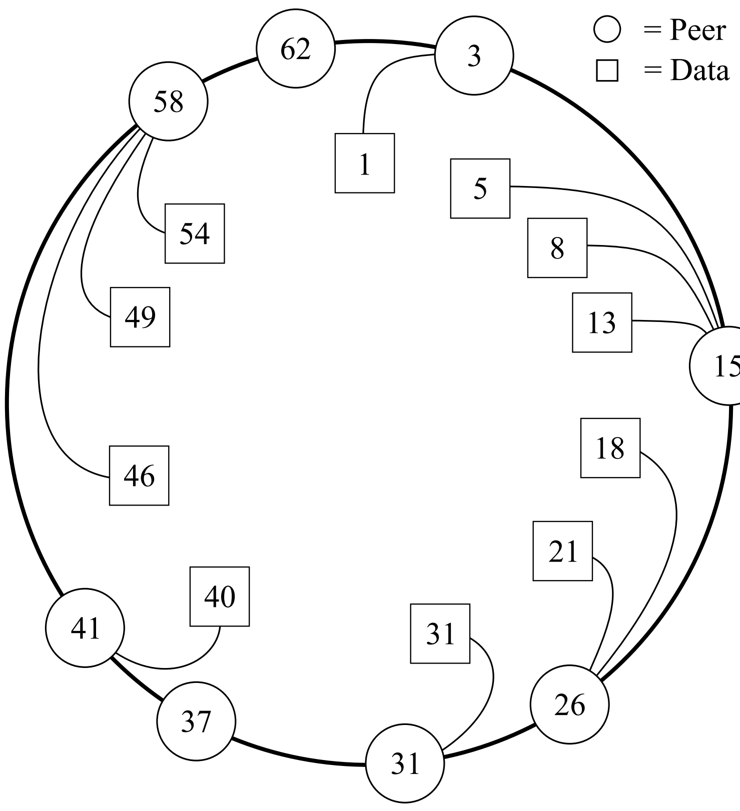


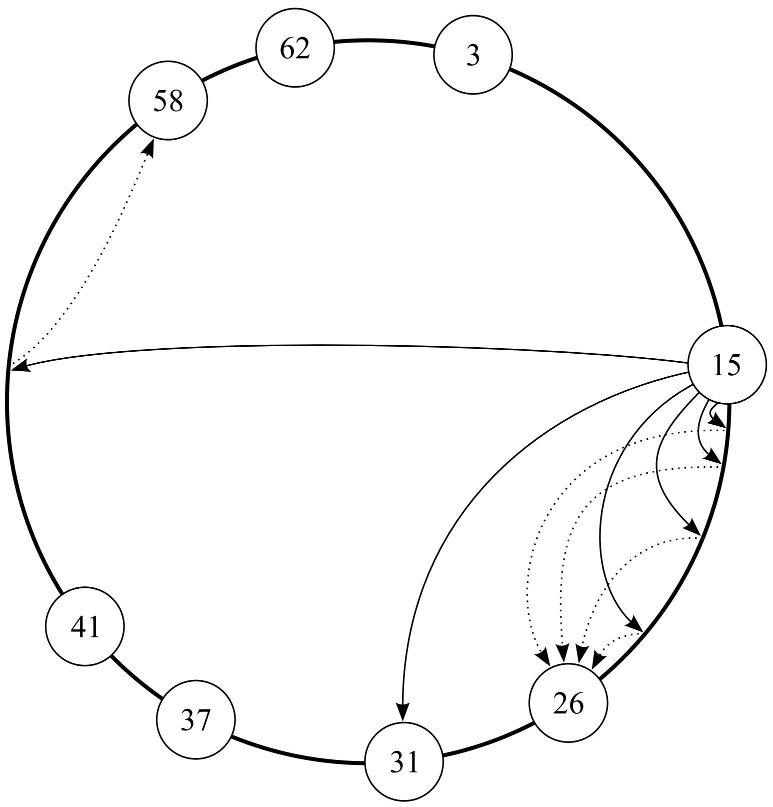
Figure 2.6: Chord Ring. This figure has been developed during the diploma thesis [Tie08]

2.2.2.1 Ring (Chord)

There exist several ring based DHT geometries such as Chord [SMK$^+$01], Symphony [MBR03], and Cycloid [SXC06]. In this section, we shall discuss only the chord DHT in detail.

One of the most used DHT geometries is the ring geometry in form of the chord DHT. Chord uses SHA-1 [oSN] hash function for assigning identifiers in range $(0, 2^m]$ to the data objects and the peers in the DHT, where m denotes the number of identifier bits. An identifier for a data object is generated by hashing the key of the data object, whereas an identifier for a peer is generated by hashing the combination of the peer's IP address and the port number.

Chord orders peers in a "modulo 2^m" identifier ring. Position of a peers in the ring is determined by the identifier of the peer. Peers are distributed almost uniformly over the ring due to the randomness of the SHA-1 hash function. A data object with an identifier i is managed by a peer that is the clockwise successor of i on the identifier ring, denoted by $successor(i)$. Figure 2.6 shows an example of a chord ring with $m = 6$. In this example, $successor(1) = 3$, $successor(5, 8, 13) = 15$, and so on.



Finger Table		
i	$\mathbf{n + 2^{i-1}}$	**ID**
1	16	26
2	17	26
3	19	26
4	23	26
5	31	31
6	47	58

Figure 2.7: Chord Ring with Finger Table. This figure has been developed during the diploma thesis [Tie08]

In a basic chord network topology, each peer has a link to the peer that follows it's identifier on the identifier ring. A link here means a mapping from peer's identifier to its IP address and port number, e.g., the link between peer 15 and peer 26 in Figure 2.6 could be stored at peer 15 in the form $26 \mapsto$ (ip = 192.168.3.5, port = 5000).

Given an identifier q, chord provides the functionality for looking up the peer that is responsible for q. Using the basic network topology, a lookup request for q is routed clockwise through the ring until $successor(q)$ is found. The basic routing strategy requires maintenance of $O(1)$ local state at each peer. However, the worst-case lookup latency is as high as $O(N)$, where N is the total number of peers in the network.

In order to reduce the lookup latency in chord, each peer maintains m number of links to other peers in the network. Out of these m links, one link connects the peer to the succeeding peer, whereas other links are typically long range links. Each link is known as a finger and the table maintaining these fingers is known as the finger table of the peer. If n is the identifier of a peer in the chord ring, then the i^{th} entry in its finger table points to $successor(n + 2^{i-1})$. Figure 2.7 shows a Chord ring with the finger table for peer 15. For simplicity, the IP addresses and the port numbers of peers have been omitted from the finger table. Since m in this example is 6, the finger table has 6 entries.

When a peer receives a lookup request for an identifier q, it forwards the request to the farthest finger that is smaller than q. This process continues through the ring until a peer p is found such that q lies between the identifier of p and the identifier of the node succeeding p. In that case, the succeeding node is responsible for q. In the example shown in Figure 2.7, if a lookup is performed for identifier 38 starting from peer 15, the request would be forwarded using the 5^{th} finger to $successor(31)$ because forwarding it to $successor(47)$ would be overshooting.

Due to the finger table, the size of the local state for each chord peer is $O(m)$. However, the worst-case lookup latency is reduced to $O(log_2 N)$ because the distance to the destination node is halved in each step during the lookup.

A new peer joins the chord ring by contacting a peer that is already a part of the ring, this process is known as bootstrapping. The joining peer finds its location in the ring by performing a lookup for its own identifier. After joining the ring, the finger table of the new peer is initialized. A peer leaves the network by informing the succeeding and the preceding peers in the ring. When a peer leaves the network, the data that it was responsible for is redistributed to the new responsible peers. Peer failures may result in loss of data which could be avoided by using a data replication strategy. For more detail on data replication in chord, see [SMK+01].

The chord ring has been designed to be efficient, scalable, and robust in order to handle the dynamic nature of a P2P system. Peers can join and leave the network at any time. For correctness of routing, the only requirement is that each peer maintains its correct successor in the identifier ring. Concurrent peer joins, departures and failures may leave the chord ring in an inconsistent state where some peers have incorrect successor and long range links. During this inconsistent state, applications may be required to retry a lookup request certain number of times before it is successfully resolved. In order to remove these inconsistencies from the chord ring, the chord protocol periodically executes a distributed self-stabilization algorithm. The details of this algorithm are beyond the scope of this discussion.

2.2.2.2 Torus/Hypercube (CAN)

The torus DHT geometry has been opted by the content addressable network (CAN) [RFH+01]. CAN is based upon a d-dimensional Cartesian coordinate space, where $d = 2 : [0, 1]$ x $[1, 0]$. The coordinate space is wrapped around the edges, hence forming a torus shape. Given an identifier q, CAN provides the functionality for looking up the peer that is responsible for managing q, where q is a point in the d-dimensional coordinate space.

Using a deterministic hash function, CAN maps each data object to a point in the d-dimensional coordinate space by hashing the key of the data object. Each data

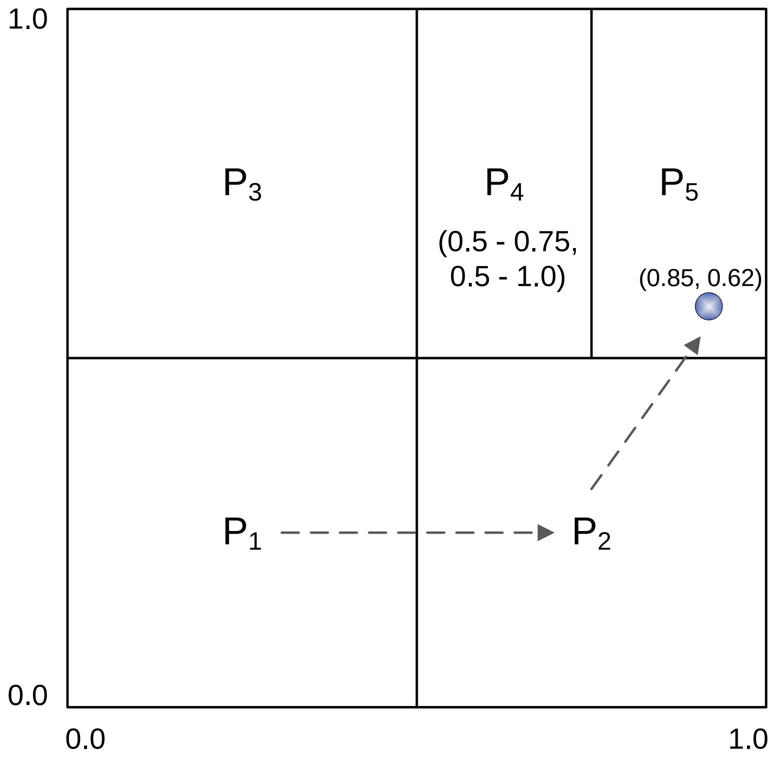


Figure 2.8: CAN Network Topology

object is managed by a peer that is responsible for a certain range of coordinates in the coordinate space. CAN partitions the d-dimensional coordinate space into zones. In the simplest form of CAN, each zone is managed by a single peer. Figure 2.8 shows a 2-dimensional CAN coordinate space with 5 peers. For better readability, the wrapping of the coordinate space around the edges is not shown in the figure. In this example, peer P_4 is responsible for data objects with coordinates in range ($[0.5, 0.75]$ x $[0.5, 1.0]$). Consequently, a data object with coordinates $(0.85, 0.62)$ would be mapped to peer P_5 as shown in the figure.

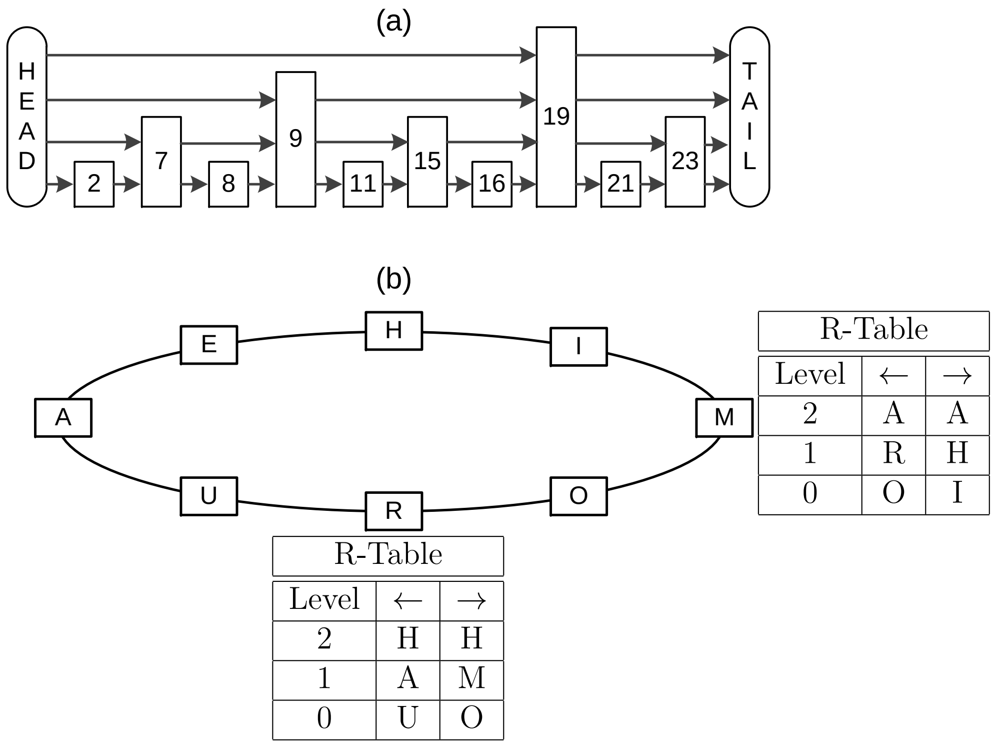
Search for an object q is carried out in an intuitively straight-forward manner, i.e., by forwarding q to the neighbour that is closest to the destination coordinates until it arrives at the peer that is responsible for managing q. Each peer in a CAN network maintains a link to its immediate neighbour. A link here is the pair of peer's IP address and zone coordinates. Two peers are neighbours if their coordinates overlap in $d-1$ dimensions and differ in 1 of the dimensions. For example, the neighbourhood of peer P_1 in Figure 2.8 includes peers P_2 and P_3 but not peer P_4 because P_4 differs P_1 in 2 dimensions. Figure 2.8 also shows how a query for an object with coordinates $(0.85, 0.62)$ is carried out if it is initiated at peer P_1.

On average, each peer in a CAN network with d dimensions and n equal zones maintains $2d$ number of neighbour links. Unlike, the chord DHT, the size of a peer's neighbourhood is independent of the total number of peers in the DHT. The average routing cost for a message in CAN is $(d/4)(n^{1/d})$ because each dimension has $n^{1/d}$ peers and the average distance to the destination along each dimension is $(1/4)(n^{1/d})$. The path length scales as $O(n^{1/d})$ with increasing number of peers, which is longer than the chord DHT.

A new peer joins a CAN network by contacting a peer that is already a part of the network. The joining peer randomly generates a point in the d-dimensional space and sends a join message to the peer that is responsible for managing that point. The contacted peer splits its zone in two halves; one half is kept by the peer itself, whereas the other half is assigned to the joining peer. A zone is split according to a pre-defined ordering of dimensions, e.g., in case of 2 dimensions, a zone is first split along x-axis, then along y-axis, and then again along x-axis and so on. The data objects that fall under the zone of the new peer are migrated to it and the neighbourhood table of each affected peer is also adjusted accordingly.

A peer can also gracefully leave a CAN network at any time. The leaving peer hands over its zone to a neighbouring peer, known as the takeover peer. The data objects belonging to the leaving peer are migrated to the takeover peer and the neighbourhood tables of the affected peers are adjusted accordingly. The takeover peer merges the two zones if a valid merge operation is possible, otherwise, it temporarily handles two zones. Furthermore, in a CAN network, each peer periodically exchanges a heartbeat message with all its neighbours. A peer is considered to be faulty by a neighbour if there is no exchange of a heartbeat message among them for a prolonged period of time. In that case, the peer that detected the faulty neighbour, removes the faulty peer from its neighbourhood table and sends a recovery message to a takeover peer. More details regarding the recovery protocol could be found in [RFH+01].

Several optimizations have also been suggested for the CAN DHT. The first optimization suggests that a high dimensional CAN should be used in contrast to a CAN with small dimensionality. Increased number of dimensions reduces the path length for routing messages at the expense of a small increase in the number of neighbours for each peer. Additionally, the increased number of neighbours improves the fault tolerance of the system because the number of alternate paths to route a message also increases. Second optimization suggest the use of multiple coordinate spaces with a peer being part of each coordinate space. Each coordinate space is then termed as a reality. Use of multiple realities improves the fault tolerance of the system because peers can route messages using alternate paths in different realities. Moreover, a data object is replicated by each reality which increases the robustness of the system. Peers can route a query to the closest replica of a searched data objects. Other CAN optimizations include construction of the CAN overlay network to reflect the underlay network, and zone overloading where multiple peers are responsible for managing a single zone increasing the robustness of the system. Several other optimizations have been discussed in [RFH+01].



R-Table		
Level	←	→
2	A	A
1	R	H
0	O	I

R-Table		
Level	←	→
2	H	H
1	A	M
0	U	O

Figure 2.9: (a)A Perfect Skip List, (b) Basic SkipNet Topology with Routing Tables

2.2.2.3 SkipList (SkipNet)

DHTs such as Chord (see Section 2.2.2.1) and CAN (see Section 2.2.2.2) distribute the data objects uniformly over a network of peers. This means that an application does not have any control over placement of its data. The data could be stored far away from the user, or even spread across several administrative domains in case of large-scale shared DHTs. These issues have been addressed by the SkipNet DHT. In addition to the general DHT functionality of efficient routing, robustness, and self-organization, SkipNet also ensures content and path locality. Content locality means that the data could be explicitly placed on certain peers in the network, e.g., on peers within a single company, whereas path locality means that the path to the local data objects stays within the local domain.

SkipNet DHT is inspired from an in-memory data structure known as the Skip List [Pug90]. A skip list is a sorted linked list of nodes where each node has several pointers pointing in the forward direction. Some of these pointers skip over some nodes in the list, hence the name skip list. The total number of pointers that a node has defines the height of the node and a pointer at level l as a length of 2^l. In a perfect skip list, the height of the i^{th} node is the exponent of the largest power of 2 that divides i. Figure 2.9(a) shows an example of a perfect skip list. Search in skip list starts at the head node. At each node, the link that arrives at a node that is closest to the searched node

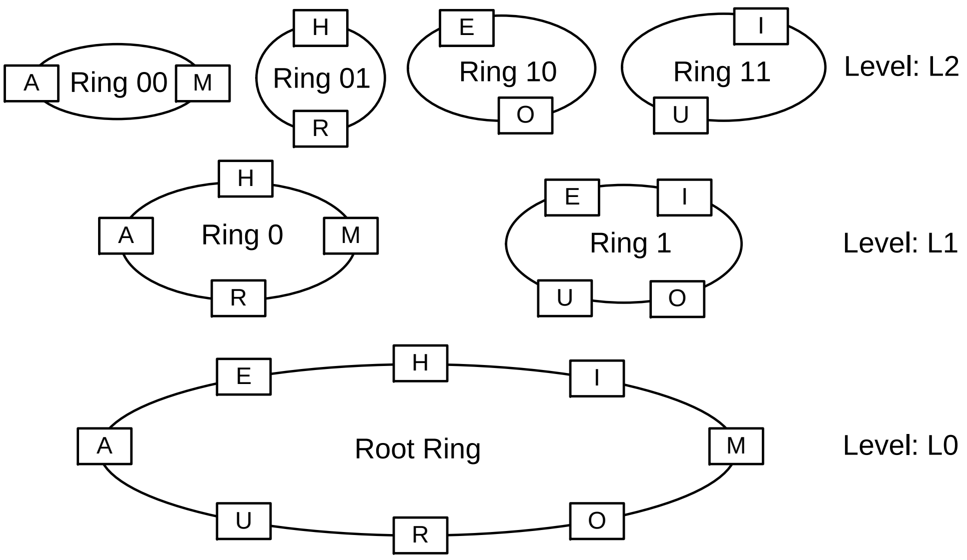


Figure 2.10: Detailed SkipNet Topology

is taken, until either the searched node is found or the search reaches the tail node. The worst-case path length for a search is $O(logN)$.

In SkipNet, the nodes from the skip list are the peers that form an overlay network. These peers are organized in a sorted ring. The position of a peer in the ring is determined by name string of the peer. Peers are sorted according to the lexicographical order of the name string. The peer name strings are not hashed values, therefore they are not uniformly distributed over the ring. The ring is doubly linked where each peer stores $2 \cdot log\ N$ links to other peers in the network. Pointers on level l of a peer point to nodes roughly 2^1 to the left and right of the peer. Figure 2.9(b) shows an example of a SkipNet where pointers of peers M and R are shown. These pointers are shown as the routing tables of the peers, also known as the "R-Table". Note that this SkipNet is a perfect one because each pointer at level l traverses exactly 2^l peers.

Pointers on each level of a SkipNet could be seen as individual rings. Figure 2.10 shows the detailed topology of the SkipNet shown in Figure 2.9(b). The ring for level $l + 1$ is obtained by splitting the ring at level l in two rings with each ring containing every second peer from the ring at level l. Since the maintenance of a perfect SkipNet is costly, a probabilistic approach is taken when splitting the ring at level l. Using this approach, a peer randomly and uniformly decides its membership to a ring at level $l+1$. This random choice of a peer could be encoded in a binary number which becomes a peer's numeric identifier. As shown in Figure 2.10, the first l bits of a peer's numeric identifier determine its membership at level l. Note that in this probabilistic SkipNet each peer still skips over 2^l nodes with high probability, whereas joining and leaving node only affects two peers in each ring.

Search in SkipNet could be carried out either using a name string or a numeric identifier.

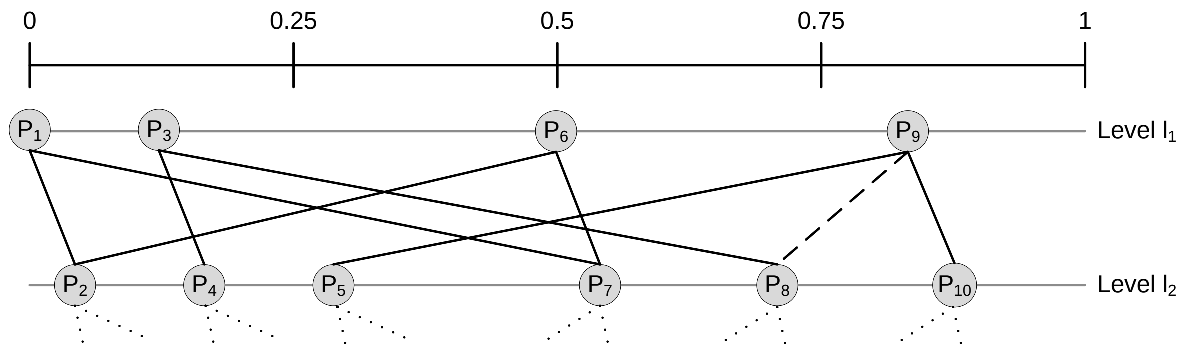


Figure 2.11: Viceroy Network Topology

If a query is performed using a name string then each peer simply follows the pointer on the highest level that does not overshoot the searched name string. Search proceeds either clockwise or anti-clockwise through the ring depending upon the proximity to the searched name string. The search terminates at a peer whose name string is closest to the searched name string. The latency of the name string based search is $O(logN)$ with high probability.

Search using a numeric identifier starts at level zero of a peer. The query is forwarded through the level zero ring until a peer is found whose numeric identifier matches the searched identifier in the first digit. The search jumps to next level ring at this peer. The search continues on this level until a peer is found whose numeric identifier matches the searched identifier in the second digit. By repeating this process, the search reaches a level l such that none of the peers at this level have $l + 1$ digits common with the searched identifier. In this case, the search ends at a peer whose identifier is numerically closest to the searched identifier. The latency of the numeric identifier based search is $O(logN)$ with high probability.

In order to place data objects at specific peers in a SkipNet, each data object's identifier is prefixed with the peer's name string that should be responsible for hosting it. Peers within the same organization should have same prefix to their name strings for the routing to stay within the same organization.

2.2.2.4 Butterfly Networks (Viceroy)

DHTs such as Chord (see Section 2.2.2.1), CAN (see Section 2.2.2.2), and SkipNet (see Section 2.2.2.3) provide logarithmic search performances. However, they require maintenance of a routing table whose size increases with increasing number of peers in the network. For a large network size, the maintenance of these links could generate a large amount of control traffic. Butterfly networks try to avoid this traffic by keeping the size of the routing table constant. Viceroy [MNR02] is one of such networks.

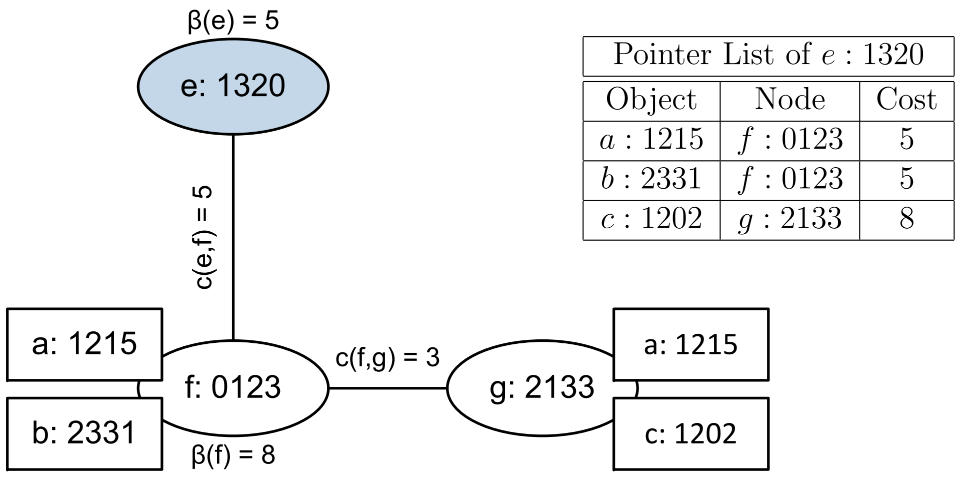
The basic topology of a viceroy network is a ring structure similar to that of a chord ring but without long distance links. Each peer has an identifier in range $[0, 1)$ chosen uniformly at random. Peers are arranged in an identifier ring where they have links to their respective predecessors and successors in the ring. Each peer is responsible for data objects with identifiers between its predecessor's identifier and its own identifier. The identifiers for the peers and the data objects are generated in the same manner as in chord.

In addition to the basic ring topology, each viceroy peer has specifically five more long distance links to other peers in the network. These links are added as follows. Each peer uniformly generates a random number in range $[1, logN]$, where N is the total number of peers in the network. This number denotes the level of the peer. Each peer at level l has two links to the level $l + 1$. The first link, known as the down-left link, connects the peer to a peer on level $l + 1$ that has an identifier that is clockwise closest to the peer's identifier. The second link, known as the down-right link, connects the peer to a peer on level $l + 1$ that has an identifier that is clockwise closest to the peer's identifier plus $1/2^l$. Additionally, each peer has a link, known as the up link, to a peer on level $l - 1$ (for $l > 1$). The up link connects the peer to a peer on level $l - 1$ that is clockwise closest to the peer's identifier. Furthermore, each peer also has two level links which connect it to the preceding and succeeding peers on the same level. Figure 2.11 shows an example of a viceroy network. For better readability, only two levels of the topology without the basic ring links are shown in this figure. At level l_1, the down and the level links of each peer are shown. At level l_2, the up-link of only peer P_8 is shown in the figure. Other up-links have been omitted to avoid clutter.

Ideally, each level in a viceroy network should have almost equal number of peers due to uniform selection of a level between 1 and $logN$ by a peer. However, the total number of peers in the network, i.e. N, is not known to each peer. Moreover, counting all peers is too expensive in a highly dynamic environment of a P2P system. Therefore, each peer estimates the value of N by measuring the distance between its own identifier and the identifier of the succeeding peer. Using this estimated N, a level is chosen by a peer between 1 and $logN$. If $logN$ changes, the peer also changes its level and updates its links accordingly.

A peer joins a viceroy network as follows. First, the peer joins the general ring in the same manner as a chord peer joins the identifier ring (see Section 2.2.2.1). After this step, the peer has its predecessor and successor links set. Next, the peer selects a level for itself using the procedure discussed above. After that, the peer finds its neighbours on the level ring by single stepping on the general ring. The peer then finds the down-left neighbour by clockwise single stepping on the general ring. The down-right neighbour is found by performing a lookup and then single stepping on the general ring. Finally, the peer finds its up neighbour also by single stepping on the general ring. Hence, all seven links are set after this procedure.

A lookup for a data object (using its key) is performed using a 3-phase query routing



Pointer List of $e : 1320$		
Object	Node	Cost
$a : 1215$	$f : 0123$	5
$b : 2331$	$f : 0123$	5
$c : 1202$	$g : 2133$	8

Figure 2.12: Plaxton Nodes and Data Objects

process in viceroy. During the first phase, the query is forwarded using the up links starting from the query initiator until the query reaches at a peer on level 1. In the second phase, the query is sent downwards through the levels using the down links. During this phase, if the query is at a peer on level i then the distance to the target is almost $1/2^{i-1}$. The down-right link is chosen at this peer if the distance to the target is $1/2^i$, otherwise the down-left link is chosen. This phase terminates when the query arrives at a peer that does not have any down links or if it overshoots the target. During the third phase, the general ring is traversed in either clockwise or anti-clockwise manner until the target peer is found. This lookup procedure is a simple one that does not lead to a logarithmic search performance. An improved lookup procedure uses the level links in the third phase to achieve $O(logN)$ path lengths with a high probability.

2.2.2.5 Tree (Plaxton Tree)

The Plaxton tree [PRR97] network architecture was originally presented as a method of accessing nearby copies of replicated data objects in a distributed systems. However, DHTs such as Pastry [RD01] and Tapestery [ZHS+04] are based upon the concepts of plaxton trees. Therefore, in this section we give a brief overview of plaxton trees.

A plaxton tree network consists of nodes and data objects with identifiers in form of m-bit strings. Each identifier is interpreted as a string of b-bit digits, where a digit has a value in range $[0, 2^b)$. For example, if $m = 10$ and $b = 2$, a node could have an identifier $x = 01\ 11\ 10\ 10\ 00$ where, $x[0] = 1$, $x[1] = 3$, $x[2] = 2$, $x[3] = 2$, and $x[4] = 0$. Each node and each data object has a unique identifier. Nodes in a plaxton network are totally ordered by a bijective function $\beta : V \mapsto \mathbb{N}$, where V is the set of nodes and $\mathbb{N}$ is the set of natural numbers. The cost of transmitting a byte from a node x to a node y is given by the cost function $c(x, y) \mapsto \mathbb{N}$. Each node x maintains a pointer list with each pointer pointing to a known copy of a data object. A pointer is of the form (o, y, k) where, o is the identifier of the data object, y is the identifier of the node hosting the data object, and k is the upper bound on the cost of reaching the data object starting from x. Figure 2.12 shows an example of a plaxton network, where the nodes are shown as ellipses and the data objects are shown as squares. The pointer list of the node $e : 1320$, is also shown in this figure.

Each node in a plaxton network maintains a primary neighbour table and a secondary neighbour table. Using the primary table, each node sustains links to other nodes with matching ID prefixes for each prefix length. The primary table of a node has, m/b rows i.e. one row for each prefix length, and 2^b columns i.e. one column for each possible digit. A table entry of a node with identifier x is of the form $primaryTable_x(i, j) = y$, where i denotes a table row and is in range $(0, m/b]$, j denotes a table column and is in range $(0, 2^b]$, and y is the node with minimum $c(x, y)$ that satisfies following conditions:

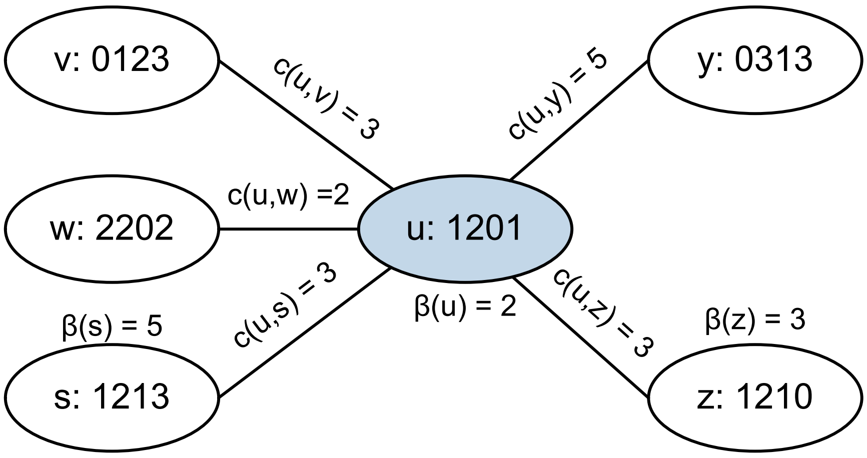
$$\begin{array}{l} \forall k < i : x[k] = y[k], \text{ and} \\ y[i] = j \end{array}$$

If no such node exists then the node y with largest $\beta(y)$ is chosen that has the most matching left-most bits with j. Figure 2.13 shows a plaxton network in which the primary table of node $u : 1201$ is shown. Details regarding the secondary neighbour table could be found in [PRR97].

In a plaxton tree, every object o has a unique root node r_o defined by its identifier. No other node has a longer matching prefix with the object identifier than the node r_o. If nodes with a same matching prefix exist, then the one with the most matching left-most bits with o becomes r_o. In case of a tie, the node with the largest β value of all nodes with equally long matching prefix becomes r_o. Node r_o knows that it is the root of o if the routing does not make any progress after m/b hops.

A node y that physically stores an object o initiates the insertion of o by forwarding an insertion message using primary neighbour path from y towards object root r_o. Each node on the primary neighbour path between y and r_o (including r_o) creates a pointer to y. A pointer is only created by a node if it does not know about any closer copy of o. If a node knows about a closer copy of o, then the message forwarding stops.

A node y searches for an object o by forwarding a search message using primary neighbour path from y towards object root r_o. The primary neighbour path is of the form $p = y, x_1, x_2, \ldots, r_o$. At each x_i, if x_i has a pointer to the requested object, it notifies the owner of the object to send a copy of o to y, otherwise it forwards the search request to x_{i+1}. At least, r_o would have a pointer to the requested object if it exists in the network. Using an optimization, each x_i might also request its secondary neighbour to return the closest copy of the object to the requester.



Primary Table of u : 1201

	0	1	2	3
x***	v : **0**123	u : **1**201	w : **2**202	w : **2**202
1**x****	s : 1**2**13	s : 1**2**13	u : 1**2**01	s : 1**2**13
12**x***	u : 12**0**1	s : 12**1**3	s : 12**1**3	s : 12**1**3
120**x**	u : 120**1**	u : 120**1**	u : 120**1**	u : 120**1**

Figure 2.13: Plaxton Node with a Primary Table

2.3 Space-Filling Curves

Typically, DHTs are capable of storing and searching 1-dimensional data based upon a single key-value pair (see Section 2.2.2). However, sophisticated application such as, an inventory keeping system, a library management system, or a file-sharing system, usually have data objects with multiple attributes (key/value pairs). These applications typically require efficient support for queries that use multiple attributes and ranges of values for searching the data. For this purpose, database management systems (DBMS) use index structures that organize the data on a storage medium in a manner that it could be accessed quickly and efficiently. One such index structure that could also be used for indexing multi-dimensional data in DHTs is the space-filling curve (SFC). A SFC has the ability to map multi-dimensional data to a line. This allows the data to be placed on a sequential storage medium, e.g., continuous blocks of a disk, or a DHT. There exist several different types of SFCs such as, Hilbert curve [Hil91], Peano curve [Pea90], Sierpiński curve [Sag94], gray curve [MAK03] etc. Figure 2.14 shows an example of a Hilber, Peano, and a Gray curve respectively. Our research mainly focuses on the use of Hilbert SFC because it exhibits best locality preserving qualities [KW06, MJFS01].

SFCs were introduced as a solution to the problem of mapping points of a line onto

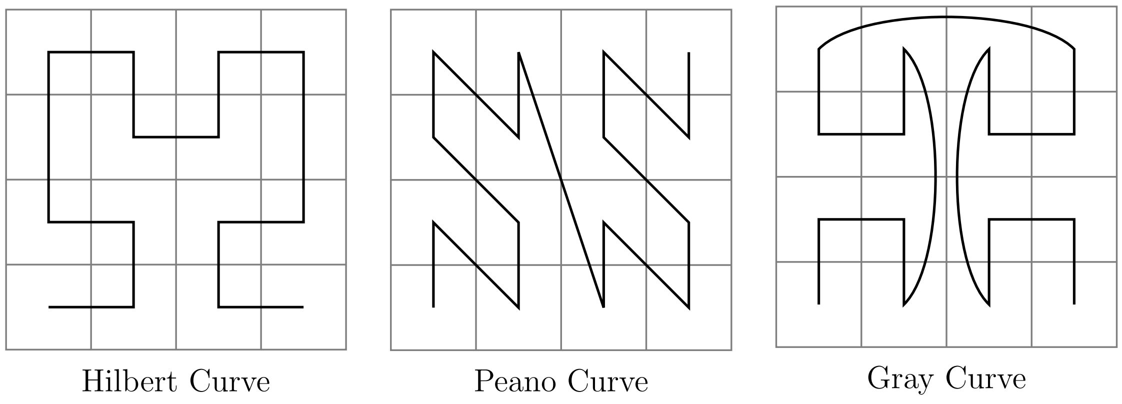


Figure 2.14: Hilbert, Peano, and Gray Space-Filling Curves. This figure has been developed during the diploma thesis [Tie08]

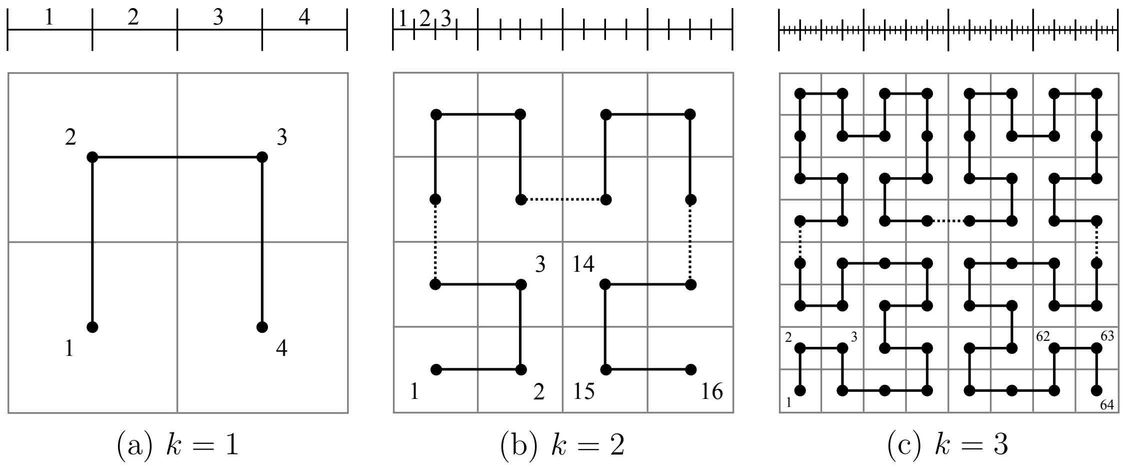


Figure 2.15: Recursive Construction of Hilbert Space-Filling Curve. This figure has been developed during the diploma thesis [Tie08]

points in a multi-dimensional space [Hil91]. Although Guiseppe Peano was the first one to solve this mathematical problem in 1890, David Hilbert was the one who a year later illustrated the solution geometrically. He showed that by dividing a 2-dimensional space into smaller sub-spaces and then allowing a curve to walk through those sub-spaces, maps the points on the curve onto the points in the 2-dimensional space, i.e., $[0, 1] \mapsto [0, 1]^2$. He also showed that the process could be inverted allowing mapping of a 2-dimensional space onto a line, i.e., $[0, 1]^2 \mapsto [0, 1]$, which is more significant for indexing multi-dimensional data.

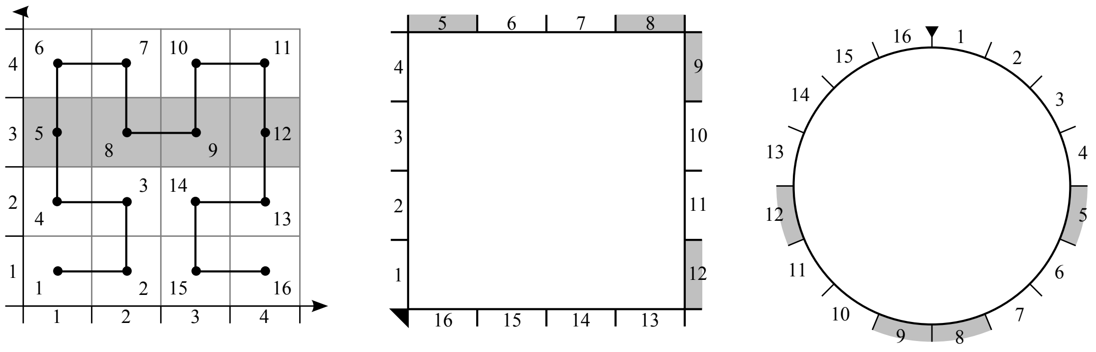
Figure 2.15 shows the recursive construction of a two-dimensional Hilbert SFC. During the first step, the two-dimensional space is equally partitioned along each dimension, and a line passes through each of the four resulting sub-spaces only once (see Figure 2.15(a)). The points in each of the sub-spaces are mapped onto a point on the line which is at the centre of each sub-space. Moreover, the points on the line are numbered in the order in which the curve moves through the two-dimensional space. As seen in Figure 2.15(a), a large number of points in a sub-space are mapped onto a single point on the line which is not a good approximation of that sub-space. Therefore, in the next step, each sub-space is further divided into four new sub-spaces (see Figure 2.15(b)). Since the new curve must pass through each of these new sub-spaces only once and it must be continuous, it is constructed using the previous curve. The upper two parts of the new curve are directly copies of the previous curve, whereas the lower two parts of the new curve are the copies of the previous curve rotated by 90° clockwise and anti-clockwise. The same process continues during the third step in which the curve is further refined to get an even better approximation of the two-dimensional space (see Figure 2.15(c)). If this process is allowed to run for an infinite number of times, the entire area of the two-dimensional space would be mapped point-by-point to the curve. The recursive construction of the curve could be extended to work on a multi-dimensional space.

The numbers used in Figure 2.15 are known as the Hilbert identifiers and they show that the process of curve construction is not confined to interval $[0, 1]$, but it can also work with integers, i.e. $\mathbf{N}^d \mapsto \mathbf{N}$. The total number of Hilbert identifiers at each approximation level k can be calculated. Consider a d-dimensional space. At the first approximation level, i.e. for $k = 1$, the space is divided into two equal halves along each dimension. For each higher approximation level until the k^{th} approximation, the total number of Hilbert identifiers is given by:

$$\begin{array}{ccccccccccc}
2_1 & \cdot & 2_2 & \cdot & \dots & \cdot & 2_{d-1} & \cdot & 2_d & = & 2^d \\
2_1^2 & \cdot & 2_2^2 & \cdot & \dots & \cdot & 2_{d-1}^2 & \cdot & 2_d^2 & = & 2^{2\cdot d} \\
 & & & & & & & & & \vdots & \\
2_1^{k-1} & \cdot & 2_2^{k-1} & \cdot & \dots & \cdot & 2_{d-1}^{k-1} & \cdot & 2_d^{k-1} & = & 2^{(k-1)\cdot d} \\
2_1^k & \cdot & 2_2^k & \cdot & \dots & \cdot & 2_{d-1}^k & \cdot & 2_d^k & = & 2^{k\cdot d}
\end{array}$$

Since each sub-space is divided into two halves along each dimension with increasing approximation level, the number of sub-spaces increases as a multiple of 2 along each dimension. The total number of sub-spaces (and therefore the number of Hilbert identifiers) is obtained by multiplying the number of sub-spaces on each dimension. This could be also written as a mapping $[2^k]^d \mapsto [2^{k\cdot d}]$, where k and d are the given approximation level and the number of dimensions respectively. It is important to note that the total number Hilbert identifiers $2^{k\cdot d}$ increases exponentially with a linear increase in the approximation level or the number of dimensions.

For using a SFC as an index over multi-dimensional data, each attribute of a data object is considered as a dimension of a multi-dimensional space. The Hilbert SFC then linearises these dimensions effectively mapping multi-dimensional data to a line. The approximation level for an SFC-based index is determined by the resolution of the attributes. If the largest range among all attributes has 100 discrete values, then the



(a) Range Query in a two-dimensional Space

(b) Mapping on the Hilbert curve

(c) Mapping on a DHT

Figure 2.16: Query Resolution using Hilbert Space-Filling Curve. This figure has been developed during the diploma thesis [Tie08]

approximation level for the SFC is given by: $\lceil log_2 100 \rceil = 7$. It should be noted that the approximation level should be the same for all dimensions. Overall, a value for the maximum approximation level depends on the application that utilizes the SFC-based index. Mostly, this number is pre-determined and fixed.

Each sub-space on a Hilbert SFC is known as a zone. A query for a specific value along each dimension of a multi-dimensional space is known as a point query. A point query maps only to a single zone on a SFC and therefore only a single Hilbert identifier is calculated by the mapping. This Hilbert identifier indicates the location of the data on the storage medium. If a query includes ranges of values along each dimension of a space, it is known as a range query. A range query usually maps to more than a single zone on a SFC and therefore multiple Hilbert identifiers are calculated for such a query. Figure 2.16(a) shows an example of a range query in a two-dimensional space. On the vertical dimension only a single section of the space is affected, whereas on the horizontal dimension all the sections of the space are affected by the query. This leads to the query being mapped to multiple zones on the SFC. These zones are 5, 8, 9, and 12. Continuous zones of a query are known as a cluster, and in this case the query maps to three clusters with two clusters containing only a single zone. Figure 2.16(b) shows the same query only on the Hilbert curve. Note that the curve is shown in a square form for a better readability. This figure indicates that the distance between the three clusters is large on the curve which means that the three different portions of the storage medium need to be accessed. The distance between zone 8 and 9 is most optimal because it allows sequential access of the storage medium which is efficient. In case the storage medium is a DHT, as shown in Figure 2.16(c), zones 8 and 9 could map to the same peer with a high probability.

SFCs are highly efficient when it comes to providing support for multi-attribute point queries. As discussed above, a multi-attribute point query results only in a single zone which is a single location on the storage medium for accessing the data. Multi-attribute range queries however map to multiple zones on a SFC which typically refer to multi locations on the storage medium. In that case, it is highly desirable that these locations are as close to each other as possible, so that the queried data could be accesses either sequentially or in a chunk. This property is known as the locality property and the SFCs try to preserve the locality of the data. It has been shown that the Hilbert SFC has the best locality preserving qualities [KW06, MJFS01].

Besides the good locality preserving quality, the Hilbert curve walks through a multi-dimensional space in a consistent manner [MAK03]. This means that it creates exactly the same number of zones in each region of the space, regardless of the dimension, i.e., no dimension is preferred by the curve. For indexing of the data, this feature is ideal because it makes the assignment of a particular attribute to a particular dimension irrelevant. Moreover, a range query in one dimension is not more cost-effective than in another dimension, because the behaviour of the curve is the same through each dimension.

Chapter 3

Optimized Information Discovery System

3.1 Introduction

Information discovery in massively distributed peer-to-peer (P2P) networks has been a widely studied topic in the last decade. The research in this area has paved the way for scalable, efficient, and fault-tolerant P2P overlay networks. One such class of scalable P2P overlay networks is the Distributed Hash Tables or DHTs (see Section 2.2.2). Conventional DHT-based overlay networks used to support data lookup using exact match queries only. However, sophisticated P2P applications require data lookup that supports more than just exact match queries, e.g., resource discovery in grid computing relies on multi-attribute range queries. Therefore, recently DHT-based systems have been extended to support a wider range of complex queries.

A fundamental class of complex queries is multi-attribute range queries. Information discovery systems such as [AX02, GYGM04, SP03, SOTZ05] make use of Space Filling Curves (SFC) [ARR+95] to support multi-attribute and range queries over DHT-based P2P overlay networks. These systems have been shown to perform well for applications that have data objects with a small number of attributes. However, the performance of such systems declines significantly with increasing number of data attributes [GYGM04, SP03, MJFS01]. In this chapter, we introduce the optimized information discovery (OID) system that does not suffer from performance deterioration with applications that have a large number of data attributes. Our system supports multi-attribute range queries by extensively optimizing the use of SFCs over DHT-based P2P networks.

If each attribute in a data object is considered as a dimension and the combination of attribute values is considered as coordinates, then each data object can be viewed as a point in a multi-dimensional space. A SFC is a line that passes through this space linearising each point in it, while preserving data locality, i.e. points close to each other in the space are mapped close on the SFC line with a high probability (see Section 2.3). The SFC line can then be mapped onto a 1-dimensional P2P overlay network such as Chord [SMK$^+$01]. Due to locality preservation, data objects that are close in the multi-dimensional space are usually mapped to sets of neighbouring peers in the P2P overlay network which allows for efficient query processing.

Currently, information discovery systems use a single SFC over all the attributes in data objects. However, in typical applications, not all attributes are used at the same time to perform a query. The attributes that are not present in the query are assumed to be wild-cards. A large number of wild-cards in a query results in a large number of segments on the SFC line. Hence, a large number of peers have to be evaluated for the query resolution. Therefore, it becomes prohibitively expensive to use SFCs in such manner for applications that a have large number of data attributes. Another concern with SFCs is that the locality preservation starts to deteriorate as the number of dimensions increases. This is due to the fact that the distance between two points in a Euclidean space usually increases with increasing number of dimensions. This property also renders SFCs unusable for applications with very large number of data attributes.

Our system design enables the efficient use of SFCs for a large number of data attributes to perform information discovery over DHTs. The basic idea is to use multiple SFC-based indices, with each index over a small number of attributes. The best SFC-based index is then selected for query processing. We also introduce two query optimizations that effectively reduce the total number of parallel messages in the network, and distribute the computation load of query resolution over the network. Our simulation results show that our query optimization strategies are more scalable than the previously suggested ones.

The rest of the chapter is organized as follows: in Section 3.2 we introduce the architecture of the OID system. Section 3.3 describes the process creating subsets of the attribute domain for the use of multiple SFC-based indices. In Section 3.4, we discuss the indexing of the data using Hilbert SFC. Section 3.5 presents the process of data placement in a DHT. Query resolution and query optimization are discussed in Sections 3.6 and 3.7 respectively. Results from the system evaluations are presented in Section 3.8. In Section 3.9 we present an overview of the related work in the area. Finally, in Section 3.10, we wind up the chapter with a conclusion and a brief overview of the work discussed in the next chapter.

3.2 System Architecture

The architecture of the OID system is a layered architecture. Figure 3.1 shows the three layers of the architecture as boxes with thick borders. The top layer is the application layer, consisting of internet-scale distributed applications such as resource discovery

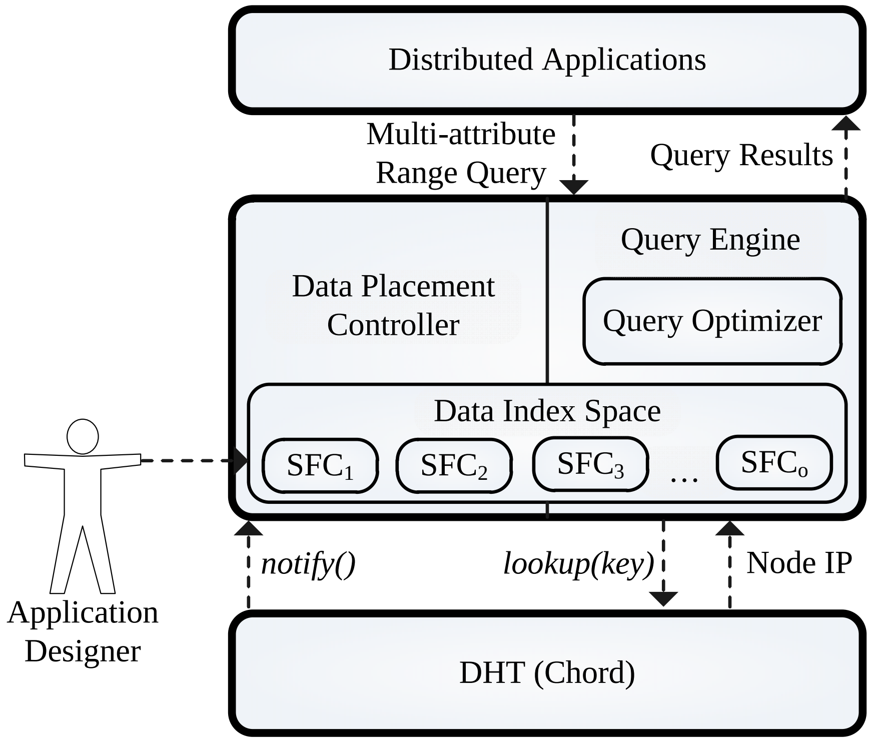


Figure 3.1: OID System Architecture

in grid computing, and location-based services. These applications query for the data using multi-attribute range queries and receive results from the layer below. The middle layer is the OID framework layer that can be further classified into three components: data index space, data placement controller and the query engine. The bottom layer is the DHT layer that given a key, returns the IP address of the peer that is responsible for hosting the data object associated with that key.

A data object in our system is represented as a list of attribute-value pairs. For example, a data object can be defined as (`CPU Speed` = 1.2 GHz, `Mem Size` = 1024 MB, `HDD Size` = 50 GB, ...). Queries are defined as conjunction of tuples of the form (attribute *operator* value), where supported operators are $=, \neq, <, >, \leq$ and $\geq$. An example of a multi-attribute range query could be (`CPU Speed` $>$ 1.0 GHz) $\wedge$ (`CPU Speed` $<$ 3.0 GHz) $\wedge$ (`Mem Size` $>$ 512 MB).

The data index space in the OID framework layer consists of multiple SFC-based indices. In order to establish the data index space, the designer of the distributed application identifies the necessary attribute combinations based on the criterion for optimizing the overall system performance. These attribute combinations are then used to define multiple SFC-based indices. Using the indices in the data index space, the data placement controller places the data objects onto the peers participating in the DHT network.

A multi-attribute range query is submitted by an application to the query engine of the OID framework layer. The query engine then uses one of the SFC-based indices in the data index space and the DHT to route the query to the peers responsible for the query resolution. The query optimizer, which is a part of the query engine achieves efficient and scalable query performance by executing two different optimization techniques.

3.3 Attribute Domain Sub-setting

As discussed in Section 3.1, information discovery systems such as [AX02, GYGM04, SP03, SOTZ05], use a single SFC-based index over the combination of all data attributes. The performance of such an index declines drastically when a query is performed only using a few attributes. Therefore, we suggest that subsets of attributes are defined such that multiple SFC-based indices could be utilized for efficient lookup.

Definition 1 *Let an attribute domain $A = \{a_1, a_2, a_3, \ldots, a_n\}$ be the set of attributes used to define data objects. Attribute domain sub-setting is defined as the creation of o sub-domains $A_1, A_2, \ldots, A_o$ such that $\forall\ i, j \in [1, o]$: $A_i \subseteq A$, $A_i \neq A_j$ for $i \neq j$, $|A_i| > 1$ and $\bigcup_{i=1}^{o} A_i = A$.*

Instead of defining a single SFC-based index for the complete attribute domain A, we define an index for each $A_i \subseteq A$. Since each SFC-based index indexes each data object, defining an index for each possible attribute sub-domain would lead to a large space requirement as well as a high index maintenance cost. Therefore, the number of attribute sub-domains is limited by the value of a parameter $o < 2^n$.

The challenge is to determine a criterion for attribute domain sub-setting such that efficient query processing is possible without defining a large number of indices. We assume a P2P information discovery system in which certain queries are much more popular than the others. This assumption is realistic, since typical P2P systems have been shown to exhibit skewed query popularity distribution [GST07, KLVW04, Sri01]. Based on this assumption, we present the following heuristic-based solution.

Let $Q = \{q_1, q_2, \ldots, q_{o-1}, q_o, \ldots, q_n\}$ be the set of queries in the system with unique attribute combinations, such that $P(q_1) \geq P(q_o) \geq P(q_n)$ for $1 \leq o \leq n$. $P(q_i)$ denotes the probability of occurrence of the query q_i. Let A_{qi} be the set of attributes used in query q_i. We define o sub-domains from domain A such that $A_i = A_{qi}$ $\forall i \in [1, o-1]$ and $A_o = \bigcup_{j=o}^{n} A_{qj}$. This heuristic creates $o - 1$ attribute sub-domains for the most popular queries and one attribute sub-domain for the rest of the less popular queries.

Although a typical P2P application uses a significant number of attributes to define data objects, it is reasonable to expect that the number of highly popular queries would still be small. Therefore, for a typical P2P application with 10 to 15 data attributes, e.g., Edutella [QNS02] that uses about 15 attributes to describe an e-learning resource, we expect o not to be larger than 15.

If the queries exhibit different popularities, then it is essential to define attribute sub-domains for the most popular queries because, if a popular query does not completely match an attribute sub-domain (hence, an SFC-based index), the accumulated overhead for such a query over a period of time, would be large. This is due to the fact that the popular queries have a higher queried frequency than the non-popular ones. Therefore, even a small overhead resulting from a popular query over a partially matching index accumulates to a large overhead over a period of time.

The attribute domain sub-setting criterion discussed above is most useful for applications where the popular queries in the system are limited and attribute combinations used in these queries could be anticipated. For applications where the query popularity could not be known in advance, more sophisticated approaches are discussed in the next chapter.

3.4 SFC-based Data Indexing

The construction of a Hilbert SFC in a d-dimensional space is a two step process (see Section 2.3). The first step divides the space into smaller sub-spaces which we call zones. The division of the space can be viewed as a recursive process where each run of the process divides the space into 2^d zones. This process continues k times resulting in $2^{k \cdot d}$ zones, where k is the approximation level. The second step involves drawing a line that passes through each of the $2^{k \cdot d}$ zones once, joining the centers of any two zones with a line segment. The centre of a zone is an approximation for all the points in that zone, and the line connecting two adjacent zones imposes an order between them. The resulting SFC is a k^{th} order SFC in a d-dimensional space. Figure 2.15 shows the 1^{st}, the 2^{nd} and the 3^{rd} order Hilbert SFCs in a 2-dimensional space.

We define an SFC-based index for each attribute sub-domain $A_i \in A$, and each SFC-based index indexes each data object in the system. As an example of resource discovery in grid computing, let $A = \{$`CPU Speed`, `Mem Size`, `Busy CPU`, `Mem Used`$\}$ be the attribute domain. If two 2^{nd} order SFC-based indices are defined for attribute sub-domains $A_1 = \{$`CPU Speed`, `Mem Size`$\}$ and $A_2 = \{$`Busy CPU`, `Mem Used`$\}$, then a data object defined as (`CPU Speed` $= 2.7\ GHz$, `Mem Size` $= 1792\ MB$, `Busy CPU` $= 63.3\%$, `Mem Used` $= 42\%$) would be indexed by the two SFC-based indices as shown with a dot in Figure 3.2. As a result, this data object is approximated by the 12^{th} and the 8^{th} zone on SFC_1 and SFC_2, respectively.

Queries with ranges over multiple attributes are mapped to blocks of zones, known as clusters. A cluster is a group of continuous zones on a SFC. The number of zones that a query would be mapped to, can be easily calculated by comparing the queried range on each query axis with the corresponding axis of the SFC. For example, a multi-attribute

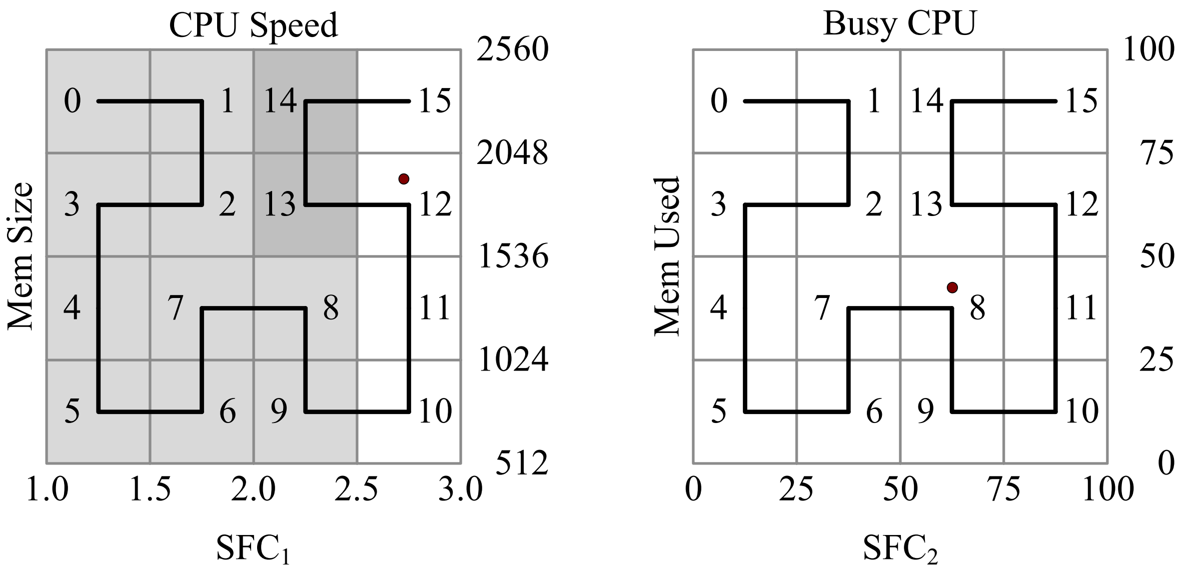


Figure 3.2: Data and Query Mapping on SFC

range query defined as "(`CPU Speed` >= 1.3) ∧ (`CPU Speed` <= 2.3) ∧ (`Mem Size` >= 640) ∧ (`Mem Size` <= 2304)", would be mapped to 12 zones on SFC$_1$ in Figure 3.2. These zones in turn represent 2 clusters, where the first cluster contains zones with identifiers from 0 to 9, and the second cluster contains zones with identifiers 13 and 14.

3.5 Data Placement

The SFC-based indices effectively map the multi-dimensional data to 1-dimensional identifiers while preserving locality. The next step involves the use of the 1-dimensional identifiers to assign the data objects to a set of peers in a DHT. We use the Chord [SMK+01] protocol as a DHT.

The Chord protocol provides the functionality for the lookup of keys in the network. Chord assigns an m bit identifier to the peers and the data objects from an identifier space in range $[0, 2^m)$. Peer identifiers are generated by hashing the IP addresses of the peers using a hash function such as SHA-1. Identifiers for data objects are generated by hashing the keys of the data objects. Chord orders the identifier space in an identifier circle modulo 2^m. A data objects is mapped to the peer whose identifier is equal to or follows the identifier of the data object in the ring (see Section 2.2.2.1).

Instead of using a hash function, an identifier for a data object in our system is generated using an SFC-based index. The peer identifiers are generated by the Chord protocol as discussed above. As an effect of the attribute domain sub-setting, the identifier space of a SFC is typically smaller than the identifier space $[0, 2^m)$ of the Chord ring. Therefore, mapping the data objects directly to the Chord ring would result in a non-uniform

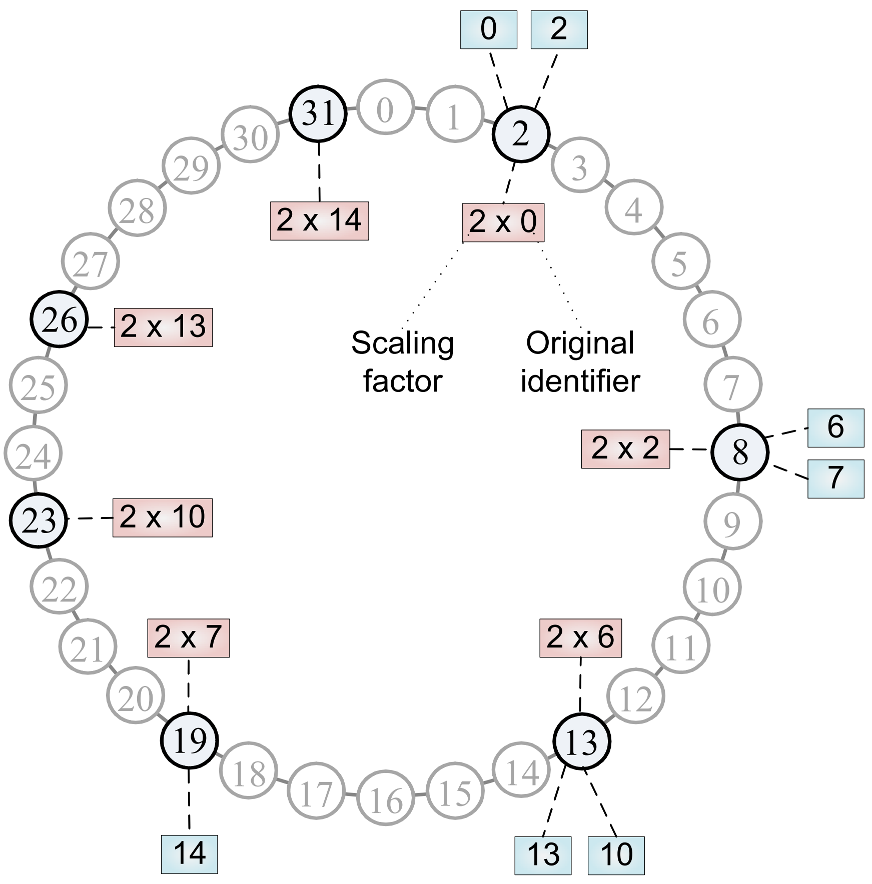


Figure 3.3: Chord Identifier Circle

distribution of the data to the Chord identifier circle. To avoid this, the identifier space of the SFC is scaled up to the identifier space of the Chord ring by a factor of $2^{m-(k \cdot d)}$.

As an example, consider a Chord ring with $m = 5$ and $N = 7$, i.e. the identifier space of the Chord ring is $[0, 32)$ with seven peers as shown in Figure 3.3. Physical nodes are shown as circles with bolder caption. Suppose an SFC-based index that assigns identifiers to the objects from an identifier space of $[0, 16)$ ($d = 2$ and $k = 2$). Without up-scaling, the data objects with identifiers $0, 2, 6, 7, 10, 13$, and 14 would be mapped to the peers as shown with the boxes outside of the Chord ring in Figure 3.3. The scaling factor is calculated as $2^{5-4} = 2$ and the same data objects with scaled up identifiers are mapped to the peers as shown with the boxes inside of the Chord ring.

Since the data index space consists of o SFC-based indices, each data object is mapped o times (possibility) to a different peer in the DHT in the similar manner as discussed above.

Up-scaling of the SFC identifier space to the Chord identifier space could result in a situation where some peers in the network are not responsible for any data objects.

This situation occurs when the number of peers is greater than the number of identifiers in the SFC identifier space. Such a situation could be avoided by setting a relatively high approximation level for the SFC-based indices.

Since the identifiers for the data objects in our system are generated using SFC-based indices, the uniform distribution of the data typically achieved by the chord hashing function is compromised. This issue is partially resolved by the use of multiple SFC-based indices; in case of our system, o number of indices. However, an explicit load-balancing algorithm might still be needed to achieve close to uniform distribution of the data. The load-balancing strategies described in [SX07] and [RLS+03] could be used for this purpose.

3.6 Query Resolution

A query for data objects can be initiated by any peer in the network through the application layer. Once a query has been initiated, the resolution is performed in four steps. In the first step, the query initiator locally selects the best matching SFC-based index that has the expected lowest overhead for the query. The second step involves calculation of the zone identifiers that match the query on the selected SFC-based index. In the third step, the identity of the peers that are responsible for the queried zones is determined and the query is forwarded to these peers. Finally, the queried peers perform a local database lookup and send the matching data objects directly to the query initiator.

When the query arrives at the OID framework layer of the query initiator, the query engine compares the attributes used in the query with the attributes used in each SFC-based index defined in the data index space. A query can either completely match a single SFC-based index or partially match more than one indices, in terms of queried attributes. If a query completely matches an index, then that index is chosen for further processing.

For partially matching queries, the goal is to select the index that uses minimum number of peers to perform a lookup. To achieve this goal, global knowledge about the mapping of zones to peers would be required. However, a peer can estimate the number of peers involved in the query resolution without this global knowledge. Let N be the total number of peers in the network. Since, the SFC identifier space is uniformly distributed over these peers, the number of zones per peer for each SFC-based index is given by $2^{k \cdot d}/N$. Let z be the number of zones that match the queried ranges on an SFC-based index. Then, the number of peers involved in the query is approximated by $(z/2^{k \cdot d}) * N$. Since the total number of peers is the same for each SFC-based index, the partially matching index with the smallest value for the fraction $z/2^{k \cdot d}$ (zone fraction) is chosen for further processing. If the zone fraction is equal, one of the indices is chosen at random.

This heuristic might underestimate the number of peers involved in a query for an SFC-based index because it assumes that all the zones on the queried peer are matching with the query. For most of the cases, the estimate is good enough because a queried peer usually stores large number of matching zones. Development of a more sophisticated heuristic that performs good estimation for all cases is discussed in Chapter 5.

The next step involves calculation of zone identifiers that match the query. If the submitted query completely matches an SFC-based index and is a multi-attribute point query, the calculation results in a single zone identifier. Querying the peer that is responsible for this zone identifier would resolve the query. For queries involving ranges of values or wild-cards, the zone identifier calculation usually results in more than one cluster with each cluster containing one or more zones.

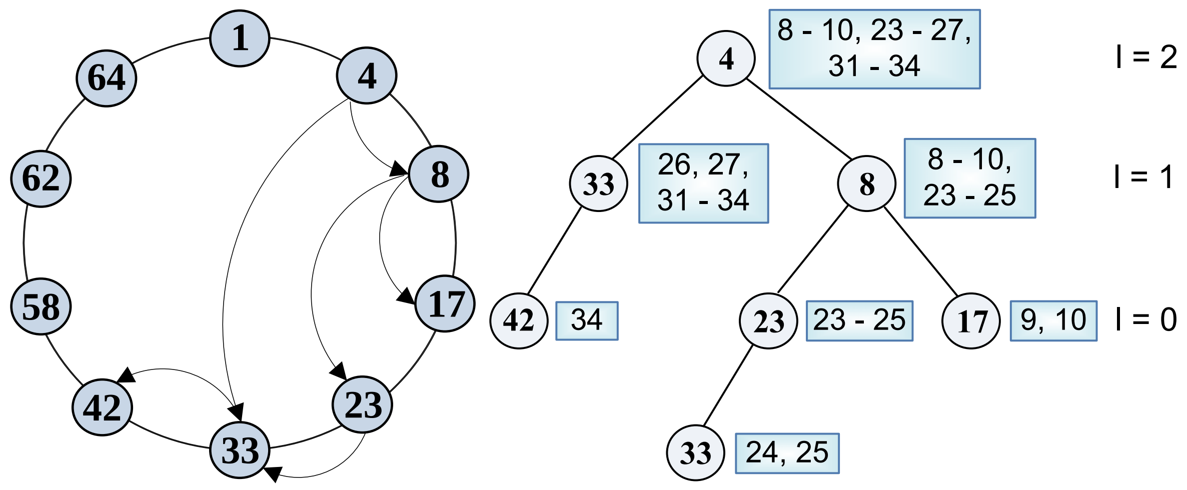
The basic approach to resolve a query that results in more than one cluster is as follows. First, a DHT lookup is performed at the query initiator for the first zone in each cluster. The DHT lookup determines the identity of the peer responsible for this zone. The query along with the cluster is forwarded to the responsible peer. If all the zones in the cluster are resolved at this peer, the cluster is completely resolved. Otherwise, this peer forwards the query to the peer responsible for the first zone in the remaining unresolved part of the cluster. This process continues until all the clusters are completely resolved. Note that the query engine has to perform the up-scaling of the queried zone identifiers before the DHT lookup could be initiated.

3.7 Query Optimization

Typically, a multi-attribute range query results in a large number of zones (and therefore clusters), whose identifiers have to be calculated. Performing this calculation at the query initiator takes a significant amount of time. Even if the zone identifiers are calculated quickly, a DHT lookup for each cluster would result in high message load at the query initiator. A peer is usually responsible for a large number of query clusters. Therefore, if the basic query resolution approach is used, the same peer could receive the same query multiple times. To avoid these drawbacks, we introduce the following two query optimizations.

3.7.1 Routing Optimization

The first optimization reduces the parallel number of messages in the network and the outgoing message load at the query initiator. The basic idea is to embed a distribution tree in the network to limit the number of messages a peer has to send, i.e., fanout of a peer. The total number of parallel messages in the network can be limited by limiting the depth-level of the distribution tree.



(a) Chord Ring with an Embedded Query Tree

(b) Routing Tree

Figure 3.4: Routing Optimization

Let parameters f and l be the fanout and the depth-level, respectively. After the zone identifiers are calculated at the query initiator, they are ordered in an ascending order and divided equally into f buckets. A DHT lookup is initiated for the first zone of each bucket. Once the peer responsible for this zone is identified, the bucket along with the query, the identity of the initiator node and the values of f and l, are forwarded to this peer. Note that, if the query initiator is responsible for some zone identifiers, it performs a local database lookup and does not include these identifiers in the buckets.

When a peer receives a bucket from some other peer in the network, it decreases the value of l by 1 and removes the zone identifiers that it is responsible for from the received bucket. The peer then performs a local database lookup. The matching data objects are directly transferred to the query initiator. If the value of l is greater than zero, the remaining zones in the bucket are further divided into f smaller buckets and each bucket is forwarded to its responsible peer. If l is zero, the remaining bucket is not divided any further and is forwarded to the peer responsible for the first zone in the bucket. In this case, the query forwarding process continues sequentially without increasing the parallel number of messages until all buckets are empty.

Consider the Chord ring with $m = 6$ and $N = 10$, shown with an embedded query tree in Figure 3.4(a). The routing tree could be separately seen in Figure 3.4(b). In this figure, peer 4 initiates a query for clusters '$8 - 10$', '$23 - 27$', and '$31 - 34$', with parameters $f = l = 2$. The peer divides the list of zones into two buckets. The bucket containing zone identifiers '$8 - 10, 23 - 25$' is sent to peer 8, because it is responsible for the first zone (8) in the bucket. The responsible peer is determined by performing a DHT lookup which has been omitted from the figure for simplicity. Similarly, the bucket containing identifiers '$26, 27, 31 - 34$' is sent to peer 33.

Peer 8 performs a local database lookup, removes zone 8 from the received bucket, divides the bucket further into two buckets, and sends them to peers 17 and 23. Equal division of the bucket at peer 8 is not possible, therefore the bucket sent to peer 23 has one more item than the one sent to peer 17, as the algorithm tries to assign continuous zones to the same buckets. The algorithm continues in a similar manner at each peer until the value of parameter l becomes zero, and there are still some unresolved zones in a bucket. This situation arises at peer 23. Therefore, the bucket at peer 23 remains undivided and is forwarded to peer 33.

The query optimization discussed above allows an application to control the message load in the network at each query level. Compared to the basic query resolution approach, the routing optimization increases the query resolution latency because fewer messages are processed in parallel. But due to the reduction in the network load, scalability is increased.

3.7.2 Computation Load Distribution

The second query optimization reduces the computation effort of calculating the zone identifiers at the query initiator. In the worst case, the query initiator calculates $O(2^{k \cdot d})$ zone identifiers, i.e., the number of zone identifier increases exponentially with d and k. The value of k is not expected to be smaller than 8 to achieve fine grained division of the SFC. The basic idea of this optimization is to distribute the computation effort of calculating the zone identifiers over several peers in the network. Once the calculation is finished, the peers that hold the calculated zone identifiers initiate the distributed query resolution using the routing optimization discussed above.

A similar query optimization has been implemented by the Squid [SP03] system. Although their optimization reduces the number of peers that are involved in the query resolution, it does not scale for popular queries in the system. In the Squid system, if a query has a high queried frequency, then the peers that perform the identifier calculations would become heavily loaded with computation tasks.

In order to distribute the computation load, we assume that each peer in the network defines a parameter $k_p \geq 1$ for each SFC-based index. k_p denotes that the maximum number of zone identifiers a peer is willing to calculate is $2^{k_p \cdot d}$. A peer first refines a query for at least one approximation level to produce zone identifiers. The peer then checks whether performing the refinement for the next approximation level would exceed the threshold of $2^{k_p \cdot d}$. If the threshold is not exceeded the peer refines the query for the next approximation level. If the threshold would be exceeded, the peer divides the identifiers into f buckets, selects f random peers in the network, and transfers the buckets to these peers. The parameter l works in the similar manner as discussed in the previous section. The process continues until the zone identifiers for the maximum approximation level k are obtained. After that, the peers that hold the final zone identifiers initiate the lookup for data objects using the routing optimization.

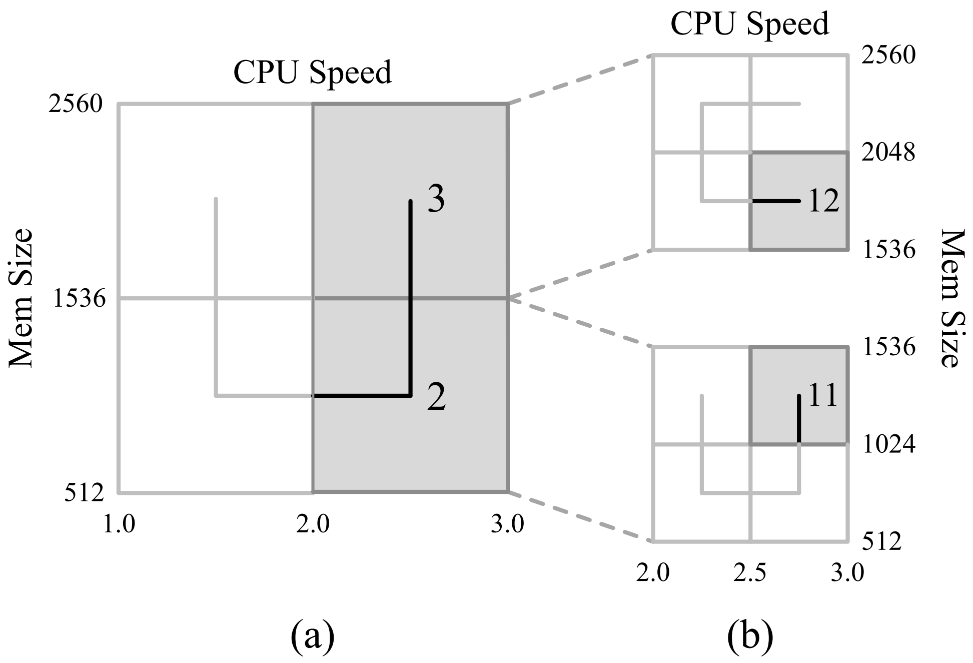


Figure 3.5: Successive Refinement of a Query

Each peer uses SHA-1 to select f random identifiers from the Chord identifier space of $[0, 2^m)$. A DHT lookup is performed for these identifiers to determine the identity of nodes responsible for them. The buckets are then transferred to these nodes for further computation. Since, SHA-1 achieves uniform distribution of identifiers in the space $[0, 2^m)$, the computation load is also uniformly distributed by each peer in the network.

For a query defined as "(`CPU Speed` >= 2.7) $\wedge$ (`CPU Speed` <= 3.0) $\wedge$ (`Mem Size` >= 1024) $\wedge$ (`Mem Size` <= 1891)" and with parameters $k = 2$, $k_p = 1$ and $f = 2$, the query initiator calculates zone identifiers 2 and 3 on the 2-dimensional SFC shown in Figure 3.5(a). The identifiers for only these two zones are calculated because others do not match the queried range. The initiator then divides these identifier into two buckets and transfers them to two random peers in the network. One of the random peer then refines zone 2 to produce zone identifier 11 and the other peer refines zone 3 to produce zone identifier 12 (see Figure 3.5(b)) for next approximation level.

For computation load distribution, each peer involved in calculating the zone identifiers has a load of $O(2^{k_p \cdot d})$. Computation load distribution also reduces the average size of the messages in the network, compared to the routing optimization. Since a peer computes only a limited amount of zone identifiers for a query, the number of clusters transmitted in a single message is therefore low.

Param	**Value(s)**	**Description**
N	$10^2, 10^{2.5}, 10^3 \ldots 10^5$	# of peers
O	$10 * N$	# of data objects
f	10	fanout
l	$\lfloor log(N) \rfloor - 1$	depth-level
k	8	k for $SFC_{1,2,3}$
k_{p1}	8	k_p for SFC_1
k_{p2}	5	k_p for SFC_2
k_{p3}	4	k_p for SFC_3

Table 3.1: Evaluation Parameters – SFC-based Indices

3.8 System Evaluation

In this section, we present experimental evaluation of several aspects of our system, using simulations. We implemented Hilbert SFC-based indices as described in [LK00, Law00] over the Chord DHT to perform these simulations.

We consider the use case of resource discovery in grid computing. An attribute domain $A = \{$`CPU Speed`, `Mem Size`, `Busy CPU`, `HDD Free`$\}$ is used to describe resources as data objects in our system. We assume that the following three attribute sub-domains have been defined by the application designer using the attribute domain sub-setting discussed in Section 3.3: $A_1 = \{$`CPU Speed`, `HDD Free`$\}$, $A_2 = \{$`CPU Speed`, `Busy CPU`, `Mem Size`$\}$ and $A_3 = \{$`CPU Speed`, `Mem Size`, `Busy CPU`, `HDD Free`$\}$. Note that the attribute sub-domain A_3 is equal to the complete attribute domain A. We define the following three SFC-based indices: SFC_1, SFC_2, and SFC_3 corresponding to A_1, A_2, and A_3 respectively.

3.8.1 Evaluating SFC-based Indices

In this section, we show that the best performance for multi-attribute range queries is achieved over corresponding completely matching SFC-based indices. Therefore, having a single large SFC is not sufficient. We also show that our system chooses the optimal SFC for query processing in case the query partially matches more than one SFCs.

For evaluating the performance of the SFC-based indices defined above, we perform 2D and 3D queries over them using the simulation parameters shown in Table 3.1. The values for f and l are chosen such that the number of parallel messages are restricted to 10% of the network size in the worst case. We increase the number of data objects with the increasing network size, just like in a typical P2P system. Also note that the value of k_{pi} is equal for the same SFC-based index at each peer in the network. The following performance metrics are measured during the simulation:

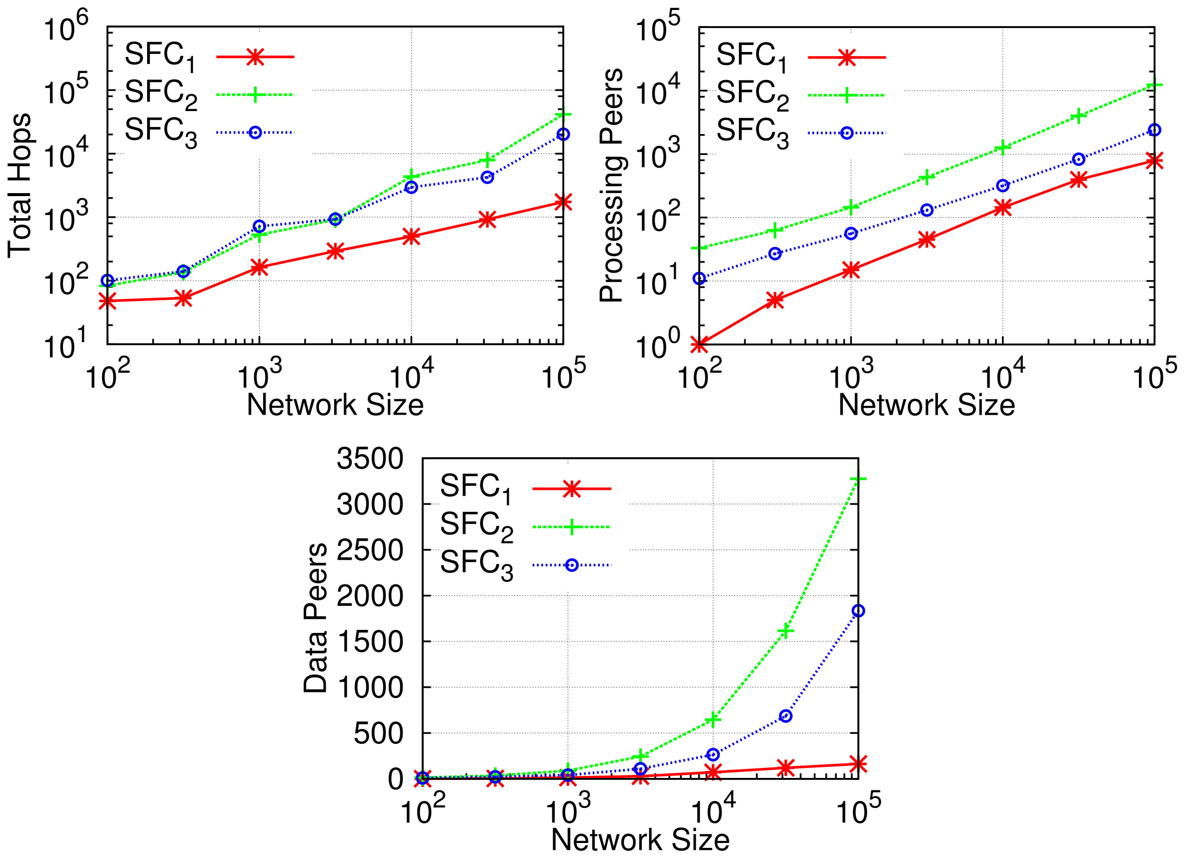


Figure 3.6: Performance of a 2D query on 2, 3 and 4 dimensional SFC-based indices

- **Total Hops** – Total number of hops of all messages needed to resolve a query.
- **Processing Peers** – Number of peers that perform a local database lookup to evaluate the query.
- **Data Peers** – Number of peers containing the data objects that match the query (subset of processing peers).

We perform a 2D range query that has the same attributes as SFC_1 and a 3D range query that has the same attributes as SFC_2, over all three SFCs. For each network size, the query is performed 10 times starting at a random peer in the network. The metrics total hops, processing peers, and data peers are then averaged over the 10 runs. The performance of the 2D query can be seen in Figure 3.6. Figure 3.7 shows the performance of the 3D query.

The 2D range query performs best on SFC_1 and the 3D range query shows best performance on SFC_2 in terms of all three performance metrics. This is due to the fact that the 2D query has the same attributes as SFC_1, and the same query has to be performed with two wild-card attributes on SFC_2 and one wild-card attribute on

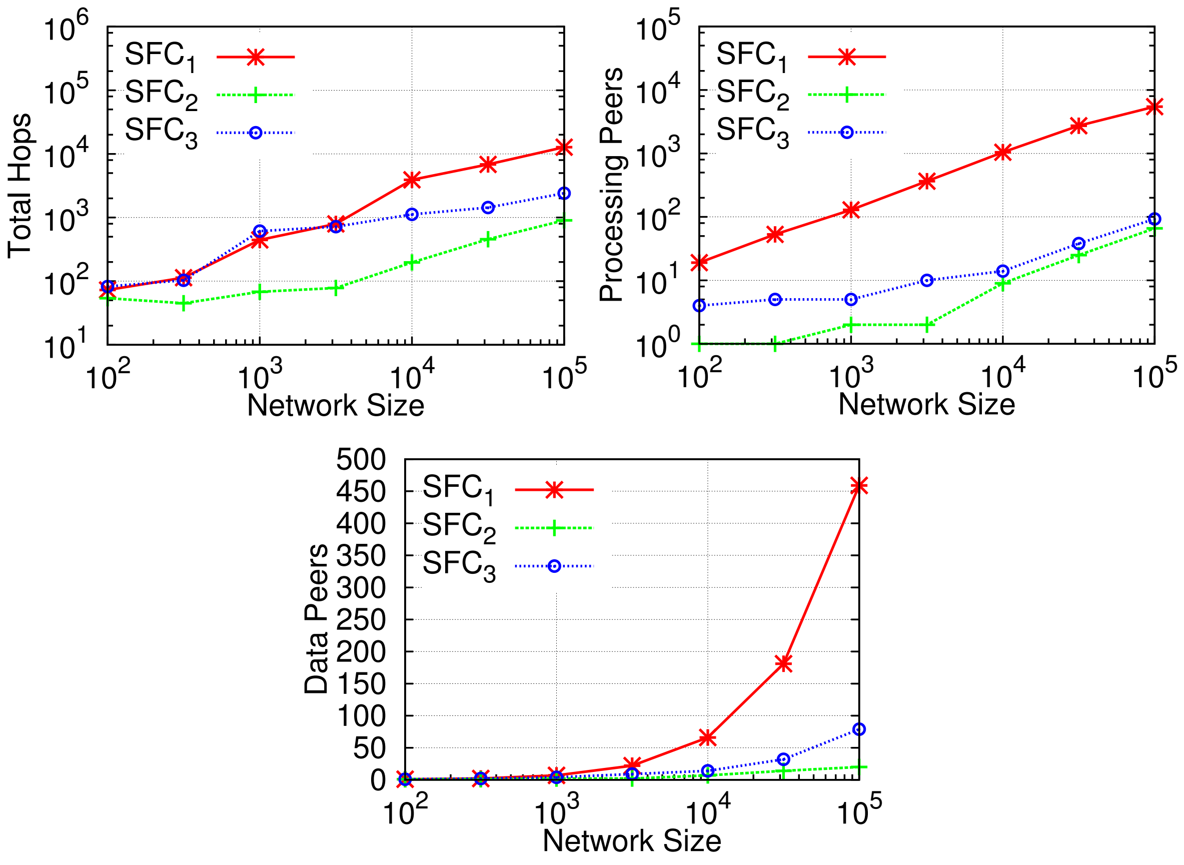


Figure 3.7: Performance of a 3D query on 2, 3 and 4 dimensional SFC-based indices

SFC_3. Similarly, the 3D query matches SFC_2 completely and has to be performed with wild-cards on the other two SFCs.

Figure 3.6 also shows that with the large network size, the 2D query becomes prohibitively expensive on SFC_2 because of the wild-cards. Similar is the case for 3D query on SFC_1 (see Figure 3.7). Therefore, it is important to perform a query using an SFC-based index that involves a minimum number of wild-cards.

If SFC_1 was not defined, then the 2D query would partially match SFC_2 as well as SFC_3. The OID system would choose SFC_3 for processing the query because the zone fraction is 0.0152 for SFC_3, compared to 0.1055 for SFC_2. Similarly, if SFC_2 was not defined, SFC_3 would be chosen for processing the 3D query because the zone fraction is 0.00042 for SFC_3, compared to 0.1055 for SFC_1. In both cases, the best matching SFCs would be chosen for the query resolution.

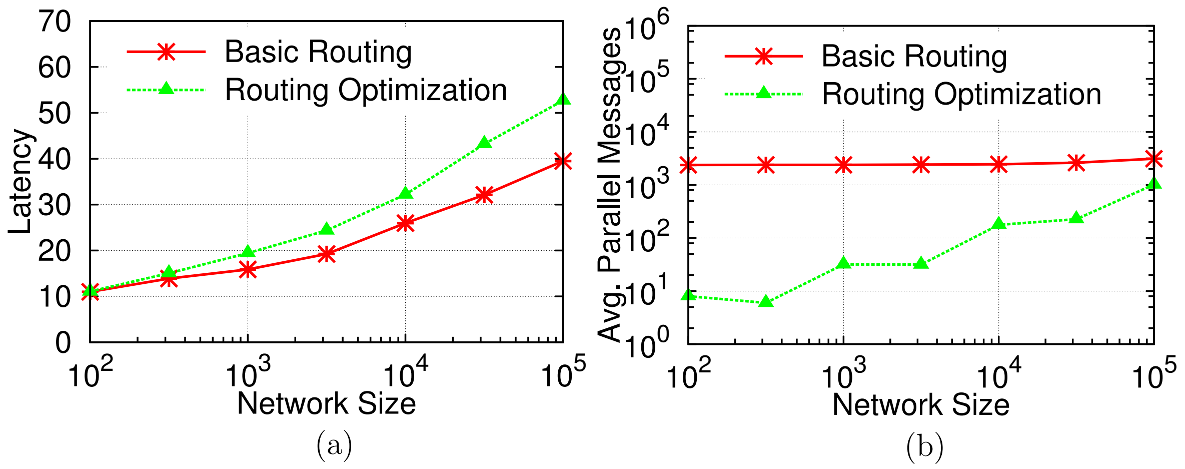


Figure 3.8: Basic Routing vs. Routing Optimization

3.8.2 Evaluating Query Optimizations

In this section, we compare the performance of the query optimization strategies discussed earlier.

3.8.2.1 Basic Routing vs. Routing Optimization

In this section, we show that the routing optimization algorithm reduces the total number of parallel messages in the network (network load) with a cost of little increased latency.

For comparison of the basic query routing strategy (see Section 3.6) with the routing optimization (see Section 3.7.1), we perform a 2D range query using attributes "`CPU Speed`" and "`Mem Size`" over SFC_2. The simulation parameters used for this experiment are shown in Table 3.2. Following performance metrics are measured:

- **Latency** – Number of hops in the longest message path.
- **Average Number of parallel messages** – Total number of forwarded messages divided by the number of levels of the search tree.

The performance of the two routing algorithms can be seen in Figure 3.8. The query resolution latency increases with the network size in case of both routing algorithms, because the number of data peers increases (see Figure 3.8(a)). As expected, the routing optimization has a higher latency than the basic approach but the difference is not major. The number of parallel messages also increase with the network size for both approaches (see Figure 3.8(b)). Even though the increase is significant for the routing optimization, the algorithm still produces fewer parallel messages per level compared to the basic approach, even with the network size of 10^5.

Param	Value(s)	Description
N	$10^2, 10^{2.5}, 10^3 \dots 10^5$	# of peers
O	$10 * N$	# of data objects
f	10	fanout
l	$\lfloor log(N) \rfloor - 1$	depth-level
k	8	k for $SFC_{1,2,3}$

Table 3.2: Evaluation Parameters – Basic Routing vs. Routing Optimization

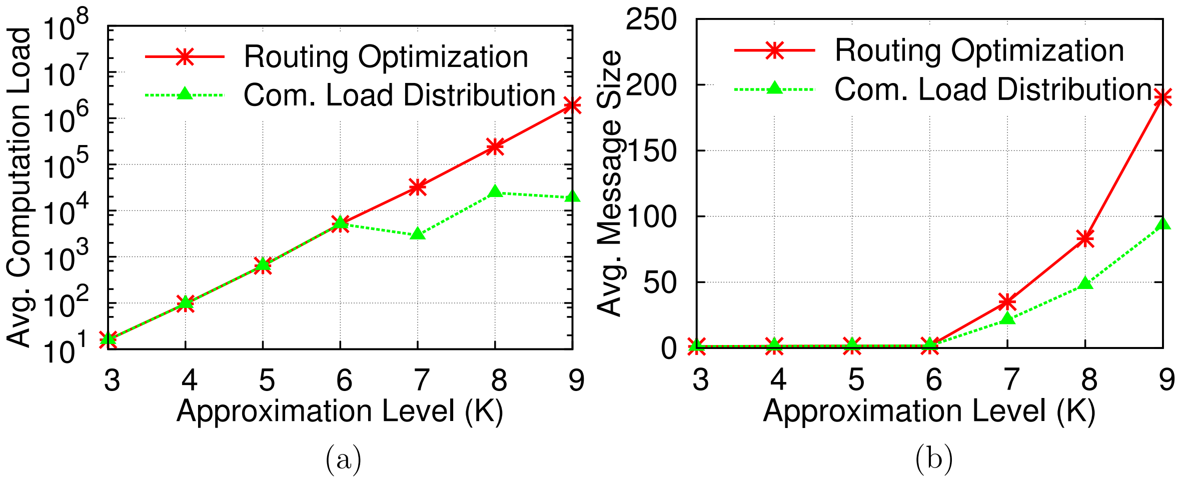


Figure 3.9: Routing Optimization vs. Computation Load Distribution

3.8.2.2 Routing Optimization vs. Computation Load Distribution

In this section, we show that the computation load of calculating zone identifiers increases exponentially with increasing approximation level k. We also show that the computation load distribution algorithm reduces the amount of computation performed by a single peer. Due to the computation load distribution, the average message size is also reduced.

For comparison of the routing optimization (see Section 3.7.1) with the computation load distribution (see Section 3.7.2), we perform a 2D range query using attributes "`CPU Speed`" and "`Mem Size`" over SFC_2. The simulation parameters used for this experiment are shown in Table 3.3. The following performance metrics are calculated:

- **Average Computation Load per Peer** – Total number of computed zones divided by the total number of peers performing the computation.
- **Average Message Size** – Total number of forwarded clusters divided by the total number of forwarded messages.

The performance comparison of the routing optimization with the computation load

Param	**Value(s)**	**Description**
N	10^3	# of peers
O	$10 * N$	# of data objects
f	10	fanout
l	$\lfloor log(N) \rfloor - 1$	depth-level
k	$3, 4, 5, \ldots 9$	k for the SFC
k_p	3	k_p for the SFC

Table 3.3: Evaluation Parameters – Routing Optimization vs. Computation Load Distribution

distribution can be seen in Figure 3.8. Note that the y-axis of Figure 3.8(a) is on a log scale.

The average computation load of peers increases with the approximation level (see Figure 3.8(a)) with or without computation load distribution, because more zone identifiers have to be calculated. However, the computation load distribution strategy reduces this load significantly, in particular for larger values of k. For example, for $k = 9$, the average computation load of a peer is reduced by 99%.

In case of computation load distribution, the same number of peers are able to calculate all zone identifiers for $k = 3 - 5$. For $k = 6$, more peers are involved in calculation, therefore the average load goes down. Similar is the case for $k = 6 - 8$.

The average message size also increases with the approximation level (see Figure 3.8(b)) in case of both optimizations, because more clusters are forwarded per message. But the computation load distribution achieves reduced message size for large values of k compared to the routing optimization. For example, for $k = 9$, the average message size of a peer is reduced by 51%.

3.9 Related Work

P2P information discovery systems that support multi-attribute range queries can be divided into two major categories. The first category includes systems that enable such queries using specialized overlay structures. Systems that stack layers on top of a DHT-based P2P overlay network fall into the second category. In this section, we briefly discuss these systems.

3.9.1 Specialized Overlay Networks

The first category of P2P information discovery systems including systems such as, distributed pattern matching system (DPMS) [AB07], Mercury [BAS04], and P-

Grid [DHJ+05], enable multi-attribute and range queries with routing guarantees by organizing peers in a specific overlay structure.

In DPMS [AB07], peers connect to form a lattice structure. Each peer can be a leaf peer or an indexing peer or both. Leaf peers advertise their content using Bloom filters [Blo70]. Indexing peers store the advertisements from the leaf peers as well as from other indexing peers. The advertisements are disseminated to a large number of indexing peers using replication which generates a large amount of advertisement traffic. This traffic is controlled by aggregating closely matching advertisements. A query is first performed using the local indices at a peer. If no matches are found, the peer sends the query to its parent. This process continues until a peer with a matching index is found. If the query reaches the highest level in the lattice without any matches, it is flooded on that level. Finally, the query descends down the lattice until a matching index is found.

In Mercury [BAS04], peers are organized into logical collections known as routing hubs. For each attribute used by the data objects, a routing hub is created. A peer can be a part of multiple routing hubs. Each hub organizes peers in a ring structure where each peer is responsible for a range of corresponding attribute values. A data object with multiple attribute/value pairs is stored at multiple corresponding hubs. A multi-attribute range query however, is carried out in a single hub using the queried range of a single attribute. The filtration of the data according to the rest of the queried attributes happens within that hub.

The P-Grid [DHJ+05] system creates a network of peers that is based on the trie structure. A trie is an ordered tree, typically used by database management systems, that allows storage and search of string based keys. Unlike a binary search tree, the strings are only stored at the leaf nodes in a trie structure. All keys that have a common prefix are stored at the same leaf node which allows efficient processing of range queries.

Due to complex structural requirements, the systems discussed above incur a high maintenance overhead. Furthermore, DHT-based overlay networks have evolved over a period of time. As a result, scalable implementations of DHTs (e.g. OpenHash [KRRS04]) are widely available which makes it easier to extend them for information discovery.

3.9.2 DHT-based Overlay Networks

Systems that enable multi-attribute range queries by stacking layers on top of a DHT-based P2P overlay network can be further divided into two classes. The first class includes systems that create an individual index for each data attribute. The second class creates a single index using the combination of all data attributes.

3.9.2.1 Individual Indices

PHT [RHRS04] uses a trie structure to enable range queries over a single attribute. Unlike the P-Grid system discussed above the trie structure is not used itself as an overlay network. Instead, a trie is constructed on top of the DHT overlay network. Multiple PHTs could be used to support multi-attribute range queries. MAAN [CFCS03] and Triantafillou et al. [TP03] use locality-preserving hashing to map the value range of an attribute on continuous nodes over the Chord ring (see Section 2.2.2.1). Multiple locality-preserving mappings are needed to support multi-attribute range queries. Andrzejak et al. [AX02] use Hilbert SFC (see Section 2.3) to map the value range of an attribute on neighbouring nodes in a CAN (see Section 2.2.2.2) network. Similarly, Shu et al. [SOTZ05] use the z-curve (aka Morton-order or z-order) [OM84] as a locality preserving mapping over Skip Graphs (see Section 2.2.2.3) to support range queries over an attribute.

All these systems resolve multi-attribute range queries in one of the following two ways: either the query is performed on a single attribute and then the data is filtered at the contacted peers, or the query is divided into multiple single-attribute range queries and each query is performed individually; later, the data is filtered at the query initiator. In the first case, the query incurs high latency due to filtering at each peer. Moreover, a large number of peers are contacted that may or may not have the data objects that match the queried ranges for all attributes. In the second case, a large number of data objects are transferred to the query initiator, only some of which match the queried ranges for all attributes.

3.9.2.2 Combined Index

SCRAP [GYGM04] uses the z-curve over Skip Graphs (see Section 2.2.2.3) to perform multi-attribute range queries. The z-curve is constructed in a multi-dimensional attribute space that represents the combination of all data attributes. The authors of SCRAP explicitly mention the problem of declining performance of SFCs with high dimensional data.

The Squid system [SP03,SP04] uses Hilbert SFC over a Chord ring to support wild-card based string queries and multi-attribute range queries. Similar to the SCRAP system, the Hilbert SFC-based mapping is defined using the combination of all attributes. Their simulations also display a similar pattern of declining performance with increasing number of wild-card attributes in multi-attribute range queries.

The ProBe [SAAA05] system extends CAN by considering the multi-dimensional data objects as points in a multi-dimensional attribute space. This logical space is then partitioned into non-overlapping rectangles with each peer being responsible for a rectangle in the overlay network. A muti-attribute range query is forwarded to the peers that are responsible for the logical rectangles that overlap with the query.

With respect to the categorization discussed above, our system is a hybrid approach to the layered DHT-based P2P systems. Similar to the first class, our system maintains multiple indices for query resolution. However, instead of having only single attribute indices, our system retains multi-attribute indices using SFCs. Each index in our system is limited in dimensionality to avoid the problems discussed in Section 3.1 with large dimensional SFCs. Similar to the second class of layered DHT-based system, our system uses a single index to resolve a multi-attribute range query. The resulting main challenge is to select the appropriate index from several multi-dimensional indices to perform the query resolution.

3.10 Conclusion

In this chapter, we presented the system design and experimental evaluation of the optimized information discovery system (OID), based on SFCs. Our system significantly improves the performance of multi-attribute range queries, particularly for applications with large number of data attributes where a single big SFC is inefficient. The optimized performance for such queries is achieved by defining multiple SFC-based indices and later selecting the best one for query processing. OID chooses the best SFC-based index for query processing by estimating the number of peers the query would be evaluated at, for each SFC-based index. We also introduced two types of query optimizations that improve the system scalability. The routing optimization limits the message forwarding load of a peer and the number of parallel messages at a time in the system. The computation load distribution algorithm distributes the computation load for the query resolution over several peers in order to avoid bottleneck at particular peers.

For the first prototype of our system, we presented a criterion for defining multiple SFC-based indices based on query popularity distribution where some queries are more popular than others. More sophisticated algorithms that work for any kind of popularity distribution are discussed in the next chapter. Furthermore, the estimation technique for selecting the optimal SFC-based index for query processing is accurate for comparison between most of the indices. However, the accuracy decreases when the difference between the compared SFC-based indices is small. Improving the accuracy of this algorithm is also discussed in the next chapter.

Chapter 4

Index Recommendation for Optimized Information Discovery

4.1 Introduction

In the previous chapter, we presented the optimized information discovery (OID) system [MTD$^+$08], which introduces a unique indexing approach for extending distributed hash tables (DHTs) to efficiently support multi-attribute range queries. The OID system creates a layer of multiple multi-attribute indices over a DHT. A multi-attribute range query is resolved by estimating the performance of the query over each index, and then selecting the one with the best estimate (see Chapter 3). Although our approach outperforms the previously proposed approaches for extending DHTs [MTD$^+$08] in terms of query overhead, the criterion for defining the initial set of indices is based on a simple heuristic (see Section 3.3). The OID system installs a user-defined number of indices for the most popular queries in the system. This criterion achieves system-wide optimal performance for applications where few popular queries make up for the largest portion of the total queries. However, for P2P applications with few popular queries and a large number of unpopular queries, this criterion would produce a set of sub-optimal indices.

Finding an optimal set of indices, irrespective of a particular type of query popularity distribution, is an NP-hard [CDN04] problem, as the number of index possibilities grows exponentially with the number of data attributes. Therefore, a heuristic-based solution that produces a close-to-optimal set of indices, is highly desirable. In this chapter, we present a set of algorithms that provides index recommendations for DHT-based information discovery systems. Given a limit for the maximum number of indices and a set of multi-attribute range queries that have been previously monitored in the system (*workload*), each algorithm recommends a set of indices that produces close-to-optimal performance for the workload within the given limit.

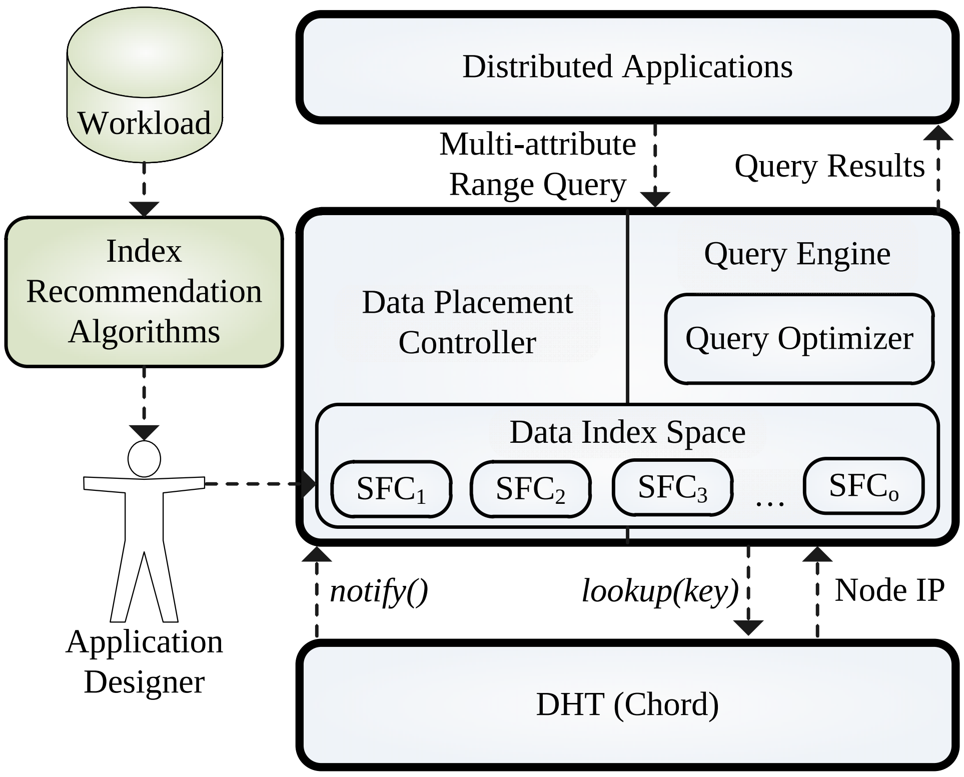


Figure 4.1: OID System Architecture

Each index recommendation algorithm, presented in this chapter, works by creating a set of candidate indices using the unique attribute combinations in the workload queries. The set of candidate indices is usually larger than the user-defined limit for the maximum number of indices. Therefore, the size of this set is successively reduced either by merging some elements or by selecting some while discarding others. Our evaluations show that in the best case, a set of indices recommended by an index recommendation algorithm is only 1.5% worse than the optimal set of indices in terms of the overhead.

The rest of the chapter is organized as follows: system architecture along with the role of the index recommendation algorithms is discussed in Section 4.2. In Section 4.3 we describe the cost estimation technique for multi-attribute range queries. The index recommendation algorithms are introduced in Section 4.4. In Section 4.5 we present the evaluation results. In Section 4.6 we give an overview of the related work. Finally, in Section 4.7 we conclude this chapter with a brief overview of the next chapter.

4.2 OID System Architecture

The index recommendation algorithms presented in this chapter provide recommendations for DHT-based information discovery systems with a 3-layer architecture discussed in the previous chapter. Figure 4.1 shows the evolution of that architecture from the one shown in Figure 3.1.

The index recommendation algorithms discussed below provide assistance to the designer of the distributed application in order to define useful multi-attribute indices for indexing the data. Each algorithm takes a limit for the maximum number of indices and a workload of multi-attribute range queries as input. It then recommends a set of indices (SFC-based indices in case of the OID system) that provides close-to-optimal performance for the workload within the given limit. This limit represents a trade-off between increased performance and index maintenance overhead.

We assume that an application-specific workload is available to the designer of the distributed application. Such a workload could be obtained either by analysing the querying trends of the application domain, or by monitoring queries in an already existing information discovery system. A discussion regarding the collection of the query workload is a part of the next chapter.

4.3 Query Cost Estimation

One of the main objectives of an index recommendation algorithm is to minimize the cost of the workload queries in a DHT network. In particular, we want to minimize the number of peers that are evaluated in order to resolve a query using an index. Given the distributed nature of P2P systems, it is extremely difficult to anticipate the exact number of peers responsible for the resolution of a query. However, a cost function can be defined that allows an index recommendation algorithm to compare different indices in order to determine the one that, with a high probability, results in the least number of peers being evaluated. In this section, we present such a cost function for the OID system based on the properties of the Hilbert SFC [Hil91].

The OID system uses Hilbert SFCs for indexing the data objects. In the context of the OID system, Hilbert SFC is defined as a continuous function $h : (a_1, a_2, \ldots, a_d) \mapsto x \in \mathbb{N}$, where $(a_1, a_2, \ldots, a_d)$ is a point in a d-dimensional euclidean space and $\mathbb{N}$ is the set of natural numbers. The process of Hilbert SFC construction divides a d-dimensional euclidean space into $2^{k \cdot d}$ sub-cubes, called zones. A line then passes through each of the zones imposing an order on them. The result is a k^{th} order SFC, where k, known as the approximation level, determines the granularity of the space sub-division (see Section 2.3).

The OID system uniformly distributes the identifier space of a SFC, $[0, 2^{k \cdot d})$, over the DHT (see Section 3.5). Therefore, with a high probability, the number of peers responsible for query resolution increases with the number of queried zones and clusters. Hence, an index recommendation algorithm can determine an index that results in the least number of peers being evaluated by calculating the number of zones and clusters for a given query over each SFC-based index. We propose the following cost function for the cost evaluation of each index:

$$cost(q) = \frac{z}{2^{k \cdot d}} \cdot c \tag{4.1}$$

where z is the total number of queried zones, d is the number of index dimensions, k is the index approximation level and c is the total number of queried clusters. This cost function is more accurate than the one presented in Section 3.6, because it also takes the number of queried clusters into account. The higher the number of queried clusters, the more the spread of the queried data objects across the DHT, and therefore the higher the cost of retrieving them.

The fraction $z/2^{k \cdot d}$, known as the zone fraction, denotes the queried proportion of the SFC-based index. Note that the total number of queried zones can be obtained without calculating the identifier for each queried zone. However, calculating the total number of clusters requires calculation of each zone identifier for each SFC-based index. Performing such a calculation could take a significant amount of time. Therefore, the actual number of clusters in Equation (4.1) is replaced by the estimated number of clusters, given by d/d_{qm}:

$$cost(q) = \frac{z}{2^{k \cdot d}} \cdot \frac{d}{d_{qm}} \tag{4.2}$$

where d is the number of dimensions of the index and d_{qm} is number of matching dimensions of the query with the dimensions of the index.

The total number of queried clusters is estimated based on the observation that the number of queried clusters increases with increasing index dimensions and decreases with each matching dimension of the query with the index. Although the exact quantitative increase or decrease in the number of clusters is not known, our simulations show that a proportional relationship is sufficient for a qualitative comparison of different indices (see Section 4.5.2).

4.4 Index Recommendation Algorithms

In this section, we discuss each index recommendation algorithm in detail. Given a workload W and a user-defined limit o for the maximum number of indices, each index recommendation algorithm searches for the o most efficient indices for the workload.

The index recommendation algorithms discussed below are independent of a particular type of cost function or indexing technique. Any type of DHT-based indexing scheme and a cost function that estimates the cost of a query, could be used in these algorithms. However, we utilize the SFC-based indices and the cost function discussed in Section 4.3 for evaluating these algorithms. Table 4.1 defines the symbols used in the discussion below.

Symbols	**Definition**
$A = \{a_1, a_2, a_3, \ldots, a_n\}$	Set of attributes used in queries
o	User-defined limit for maximum number of recommended indices
$W = \{q_1, q_2, q_3, \ldots, q_m\}$	Set of representative queries. Also known as the workload
$minCost(q, I)$	Returns the minimum cost of a query q on a set of indices I
$attributeCombination(q)$	Returns the combination of attributes used in a query q
$createSFCIndex(x)$	Returns an SFC-based index created using the attribute combination x

Table 4.1: Definition of Symbols

4.4.1 Naïve Index Recommendation

The most straight-forward (and therefore the naïve) way of determining the optimal set of indices for a workload W is to enumerate all o-sized combinations of the power-set of A, and then choose the combination with the least total cost for the queries in W. We call this algorithm the naïve index recommendation algorithm.

Although the naïve index recommendation algorithm shows an exponential growth in the complexity, it serves as a reference for the scalable index recommendation algorithms discussed in the next section. Algorithm 1 shows the pseudo-code for the naïve index recommendation algorithm.

The first step of the naïve index recommendation algorithm creates a power-set P of the attribute set A, by invoking the method $Pow(A)$ (Algorithm 1: line 1). In the next step, a set C of all o-sized combinations of P is created (Algorithm 1: line 2). Each element of this set represents a possibility for the final solution. Lines 4 to 11 of the algorithm create a set of multi-attribute indices I using each element of C and select the I that produces the least cost for the queries in W. The final step of the algorithm returns this set as the final set of recommendations (Algorithm 1: line 12).

If n_a is the size of the attribute set A, then the size of the power-set of A is 2^{n_a}. Since C is the set of all o-sized combinations of the power-set of A, the complexity of line 2 of the naïve algorithm is:

$$O\left(\binom{2^{n_a}}{o}\right), \text{ i.e., } O\left(\frac{2^{n_a}!}{o!(2^{n_a}-o)!}\right)$$

Moreover, the complexity of line 6 of the algorithm is calculated as:

$$O\left(\binom{2^{n_a}}{o} \cdot |W| \cdot o\right)$$

Algorithm 1 Naïve Index Recommendation Algorithm

```
1: $P = Pow(A) - \{\}$;
2: $C = \{c_1, c_2, c_3, \ldots, c_l\} : c_i \subseteq P, |c_i| = o, c_i \neq c_j \ \forall i \neq j$;
3: $minTotalCost = +\infty$; $totalCost = 0$; $R = I = \emptyset$;
4: for all $c_i$ in $C$ do
5:     $I = \bigcup_{p \in c_i} createSFCIndex(p)$;
6:     $totalCost = \sum_{k=1}^{|W|} minCost(q_k, I)$;
7:     if $totalCost < minTotalCost$ then
8:         $minTotalCost = totalCost$;
9:         $R = I$;
10:    end if
11: end for
12: return $R$;
```

because the loop statement (Algorithm 1: line 4) executes $\binom{2^{n_a}}{o}$ times and calculates the cost of all queries in W over a set of o indices in the worst case.

The overall worst-case complexity of the naïve index recommendation algorithm is:

$$O\left(\binom{2^{n_a}}{o}\right) + O\left(\binom{2^{n_a}}{o} \cdot |W| \cdot o\right)$$

Since the algorithm has an exponential growth in complexity, it does not scale for large attribute sets. However, it serves as a reference for the scalable index recommendation algorithms discussed in the next section.

4.4.2 Scalable Index Recommendation

The naïve index recommendation algorithm discussed above does not scale due to combinatorial explosion. The combinatorial explosion occurs mainly because of two reasons: first, the calculation of the power-set of A grows exponentially with the number of attributes (Algorithm 1: line 1) and second, the initial search space of the algorithm includes all o-sized combinations of the power-set of A (Algorithm 1: line 2).

In this section, we present three scalable index recommendation algorithms that deal with the complexity of the problem in two steps. The first step of each algorithm considers a limited set of attribute combinations as the initial search space. We call this set as the candidate set. The second step of each algorithm uses a certain heuristic to reduce the size of the candidate set to the user-defined limit o.

Given a workload W, the candidate set C is defined as:

$$C = \bigcup_{i=1}^{|W|} attributeCombination(q_i) - \bigcup_{i=1}^{|A|} \{a_i\}$$

Unlike the naïve recommendation algorithm, the size of the initial search space is now limited by the size of the workload. The set containing the combination of all attributes is removed from C because the final solution of each scalable index recommendation algorithm always includes an index created from the combination of all attributes in A. This index acts as a fall-back index for queries that could not be optimized.

The assumption behind the creation of the candidate set is that the workload does not contain queries using all possible combinations of the attributes in A. This assumption is realistic for typical P2P applications, where queries with certain attribute combinations are frequently issued while queries with other attribute combinations are almost never used [GST07, KLVW04, Sri01].

Since C includes all the unique attribute combinations in W, it also represents the optimal set of indices for the queries in W. However, typically the size of C is greater than the user-defined limit o. Any reduction in the size of C would worsen the performance of W over the set of indices created using the attribute combinations in C. Therefore, the aim of the following index recommendation algorithms is to keep the performance deterioration of W minimal, while reducing the size of C to o. In contrast to the naïve index recommendation algorithm, these algorithms are heuristic solutions that trade-off runtime for optimality.

4.4.2.1 Cost-based Merge Algorithm

The cost-based merge algorithm recommends a set of multi-attribute indices by merging pairs of attribute combinations in the candidate set C until the size of C is reduced to $o - 1$. Since merging any pair of combinations in C would increase the total cost of the workload on C, the idea is to merge the pairs that result in the least cost increase. Algorithm 2.1 and Algorithm 2.2 show the pseudo-code for the cost-based merge algorithm.

The first few steps of the algorithm select a pair of elements in C, remove the selected pair from a copy of C (*tempC*), and add the union of the pair to it (Algorithm 2.1: line 5–7). Next, a set of SFC-based indices is created using the modified copy of C and the cost of the workload is calculated over it (Algorithm 2.1: line 8–9). These steps are repeated until a pair of elements in C that results in the least workload cost is located (Algorithm 2.1: line 10–13). The pair is then removed from C and the union of the pair is added to it, making the changes to C permanent (Algorithm 2.1: line 15). The merging process repeats itself as long as the size of C is greater than $o - 1$. Once the size of C is less than or equal to $o - 1$, the algorithm creates a set of indices R from C

Algorithm 2.1 Cost-based Merge Algorithm

```
1: $R = \emptyset$; $\acute{C} = C$;
2: while $|C| > o - 1$ do
3:   $minTotalCost = +\infty$; $totalCost = 0$;
4:   $x = y = -1$; $I = \emptyset$;
5:   for all $c_i, c_j \in C : i \neq j$ do
6:     $tempC = C$;
7:     $tempC = tempC - \{c_i\} - \{c_j\} \cup \{c_i \cup c_j\}$;
8:     $I = \bigcup_{c_k \in tempC} createSFCIndex(c_k)$;
9:     $totalCost = \sum_{l=1}^{|W|} minCost(q_l, I)$;
10:    if $totalCost < minTotalCost$ then
11:      $minTotalCost = totalCost$;
12:      $x = i$; $y = j$;
13:    end if
14:  end for
15:  $C = C - \{c_x\} - \{c_y\} \cup \{c_x \cup c_y\}$;
16: end while
17: $R = \bigcup_{c_i \in C} createSFCIndex(c_i)$;
18: $R = R \cup createSFCIndex(A)$;
19: if $|R| == o - 1$ then
20:   $R = addMissingIndex(R, C, \acute{C})$;
21: end if
22: return $R$;
```

and adds an index created from the combination of all attributes to R (Algorithm 2.1: line 17–18).

Typically, R at this point in the algorithm, represents the final set of recommended indices, but it is possible that the size of C had been reduced to $o - 2$ by the previous steps of the algorithm and therefore the size of R is $o - 1$. This can happen in the case where merging a pair of elements in an o-sized C reduces C by two elements. Consider an example scenario where $A = \{a_1, a_2, a_3, a_4\}$, $o = 5$ and C consists of 5 elements shown in Figure 4.2(a). Now suppose that elements e_1 and e_3 are selected from C for the merge operation by the algorithm. After merging the selected elements, C is reduced to 3 elements shown in Figure 4.2(b). Therefore, the set of indices R, shown in Figure 4.2(b), contains 4 elements which is less than the user-defined limit of 5 indices.

After merging pairs of elements in C, if the size of R is $o - 1$ (Algorithm 2.1: line 19), the algorithm invokes the procedure shown in Algorithm 2.2. This procedure adds an

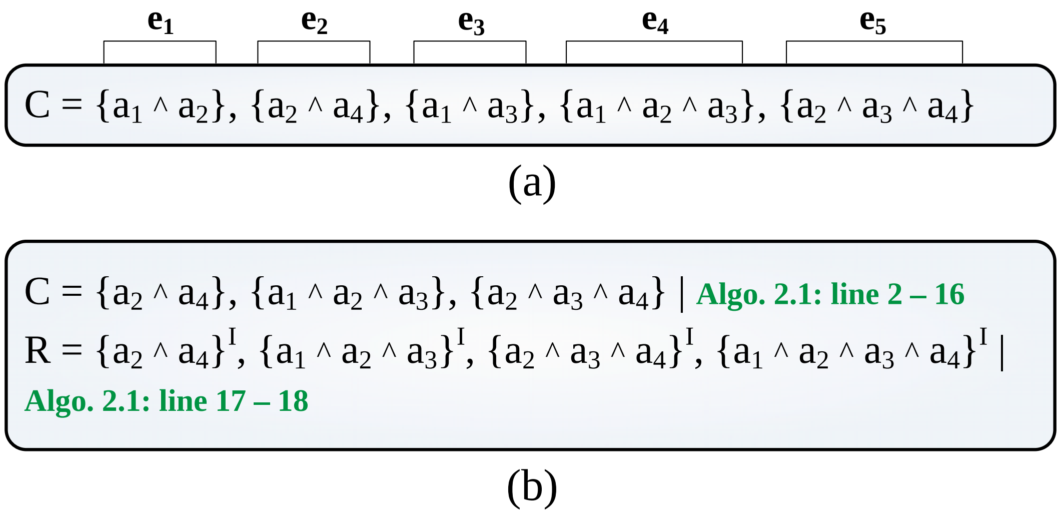

e1
e2
e3
e4
e5
C = {a1 ∧ a2}, {a2 ∧ a4}, {a1 ∧ a3}, {a1 ∧ a2 ∧ a3}, {a2 ∧ a3 ∧ a4}
(a)
C = {a2 ∧ a4}, {a1 ∧ a2 ∧ a3}, {a2 ∧ a3 ∧ a4} | Algo. 2.1: line 2 – 16
R = {a2 ∧ a4}I, {a1 ∧ a2 ∧ a3}I, {a2 ∧ a3 ∧ a4}I, {a1 ∧ a2 ∧ a3 ∧ a4}I | Algo. 2.1: line 17 – 18
(b)

Figure 4.2: Execution of Cost-based Merge Algorithm

index created using an element from the original candidate set to R. A copy of the original candidate set denoted by $\acute{C}$ is provided as a parameter to the procedure. An index created using an element of $\acute{C}$ is added to R and the cost of the workload is calculated over the modified R (Algorithm 2.2: line 4–6). This step is repeated for all the elements of $\acute{C}$ and the index that results in the least workload cost over R is marked (Algorithm 2.2: line 7–10). Finally, the marked index is permanently added to R (Algorithm 2.2: line 13) and R is returned as the final set of recommended indices (Algorithm 2.2: line 14).

The worst-case complexity of the cost-based merge algorithm can be determined as follows. If n_c is the size of the candidate set C, then the first loop of the cost-based merge algorithm performs $n_c - o$ executions at most (Algorithm 2.1: line 2). The second loop of the algorithm (Algorithm 2.1: line 5) is realized as a nested loop. Therefore, it executes $\frac{(n_c{}^2 - n_c)}{2}$ times, in the worst case. The third loop of the algorithm is executed while calculating the total cost of the workload over n_c indices in the worst case (Algorithm 2.1: line 8). Hence, the complexity of line 8 of Algorithm 2.1 is calculated as:

$$O\left((n_c - o)(\frac{(n_c{}^2 - n_c)}{2})(|W| \cdot n_c)\right), \text{ i.e., } O\left(n_c{}^4\right)$$

Moreover, the complexity of line 6 of Algorithm 2.2 is calculated as $O\left(n_c \cdot |W| \cdot o\right)$. Therefore, the overall worst-case complexity of the cost-based merge algorithm is:

$$O\left(n_c{}^4\right) + O\left(n_c \cdot |W| \cdot o\right)$$

Algorithm 2.2 *addMissingIndex*$(R, C, \acute{C})$

```
1:  $minTotalCost = +\infty$; $totalCost = 0$;
2:  $A = B = \emptyset$;
3:  for all $c_i \in (\acute{C} - C)$ do
4:      $A = createSFCIndex(c_i)$;
5:      $R = R \cup A$
6:      $totalCost = \sum_{j=1}^{|W|} minCost(q_j, R)$;
7:      if $totalCost < minTotalCost$ then
8:          $minTotalCost = totalCost$;
9:          $B = A$;
10:     end if
11:     $R = R - A$;
12: end for
13: $R = R \cup B$;
14: return $R$;
```

4.4.2.2 Similarity-based Merge Algorithm

The similarity-based merge algorithm also uses merging of elements in C to recommend a set of multi-attribute indices. Pairs of attribute combinations that are most similar to each other, in terms of attributes, are merged until the size of C is reduced to $o - 1$. Compared to the cost-based merge algorithm, the similarity-based merge algorithm has reduced complexity because the elements in C are merged without actually checking them against the workload. The similarity-based merge algorithm is based on the observation that typically merging two almost identical indices will not decrease the performance of the workload significantly. Algorithm 3 shows the pseudo-code for the similarity-based merge algorithm.

The similarity-based merge algorithm starts by selecting a pair of attribute combinations in C and calculating the difference in the attributes of the pair (Algorithm 3: line 5–6). These steps of the algorithm are repeated for all pairs in C and the pair with the least difference in attributes is marked (Algorithm 3: line 7–10). The marked pair is then removed from C and the union of the pair is added to it (Algorithm 3: line 12). The algorithm keeps repeating as long as the size of C is greater than $o - 1$. Once the size of C is less than or equal to $o - 1$, the algorithm creates a set of indices R from the modified C and adds an index created from the combination of all attributes to R (Algorithm 3: line 14–15).

Consider an example scenario where $A = \{a_1, a_2, a_3, a_4\}$, $o = 4$ and the initial candidate set C consists of 5 elements shown in Figure 4.3(a). The similarity-based merge algorithm starts by merging elements e_1 and e_2 in C because the difference between the two elements is minimum (single attribute). By merging e_1 and e_2, the size of C is reduced by a single element and the modified C is shown in Figure 4.3(b). Since the size of C is still greater than $o - 1$, elements e_1 and e_3 in Figure 4.3(b) are merged to get the $o - 1$-sized C shown in Figure 4.3(c). Finally, the set of indices R shown in Figure 4.3(c) is created using the attribute combinations in C and a combination with all the attributes in A.

Algorithm 3 Similarity-based Merge Algorithm

```
1:  $R = \emptyset$; $\acute{C} = C$;
2:  while $|C| > o - 1$ do
3:      $minMergeDiff = +\infty$; $mergeDiff = 0$;
4:      $x = y = -1$;
5:      for all $c_i, c_j \in C : i \neq j$ do
6:          $mergeDiff = |(c_i \cup c_j) - (c_i \cap c_j)|$;
7:          if $mergeDiff < minMergeDiff$ then
8:              $minMergeDiff = mergeDiff$;
9:              $x = i;\ y = j$;
10:         end if
11:     end for
12:     $C = C - \{c_x\} - \{c_y\} \cup \{c_x \cup c_y\}$;
13: end while
14: $R = \bigcup_{c_i \in C} createSFCIndex(c_i)$;
15: $R = R \cup createSFCIndex(A)$;
16: if $|R| == o - 1$ then
17:     $R = addMissingIndex(R, C, \acute{C})$;
18: end if
19: return $R$;
```

Analogous to the cost-based merge algorithm, the size of the set of indices R produced by the similarity-based merge algorithm could be $o - 1$. The missing index is added to R in the similar manner, as in the cost-based merge algorithm, i.e., by invoking the procedure shown in Algorithm 2.2 (Algorithm 3: line 16–17).

If n_c is the size of the candidate set C, then the first loop of the similarity-based merge algorithm performs $n_c - o$ executions and the second loop performs $\frac{({n_c}^2 - n_c)}{2}$ executions in the worst case (Algorithm 3: line 2 & 5). Therefore, the worst-case complexity of line 6 of the algorithm is $O\left({n_c}^3\right)$. Moreover, as established earlier, the complexity of Algorithm 2.2 is $O\left(n_c \cdot |W| \cdot o\right)$. Therefore, the overall worst-case complexity of the similarity-based merge algorithm is:

$$O\left({n_c}^3\right) + O\left(n_c \cdot |W| \cdot o\right)$$

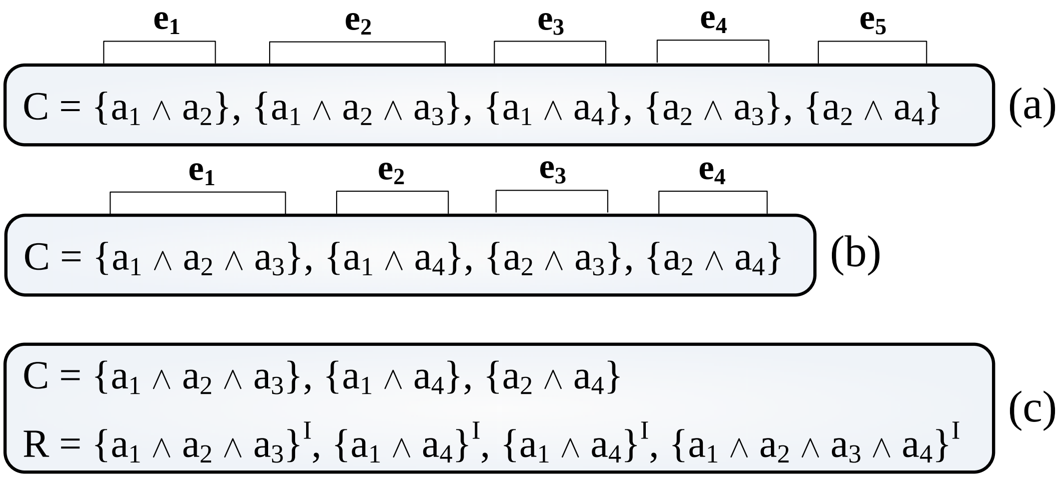

e1
e2
e3
e4
e5
C = {a1 ∧ a2}, {a1 ∧ a2 ∧ a3}, {a1 ∧ a4}, {a2 ∧ a3}, {a2 ∧ a4}
(a)
e1
e2
e3
e4
C = {a1 ∧ a2 ∧ a3}, {a1 ∧ a4}, {a2 ∧ a3}, {a2 ∧ a4}
(b)
C = {a1 ∧ a2 ∧ a3}, {a1 ∧ a4}, {a2 ∧ a4}
R = {a1 ∧ a2 ∧ a3}^I, {a1 ∧ a4}^I, {a1 ∧ a4}^I, {a1 ∧ a2 ∧ a3 ∧ a4}^I
(c)

Figure 4.3: Execution of Similarity-based Merge Algorithm

4.4.2.3 Selection Algorithm

The selection algorithm for multi-attribute index recommendation calculates the cost of the workload for each element of the candidate set C and chooses $o - 1$ elements with the least cost. The idea behind the algorithm is that if the selected elements have the least cost for the workload individually, then the probability that they have least cost for the workload altogether is also high. Algorithm 4 shows the pseudo-code for the selection algorithm.

Algorithm 4 Selection Algorithm

```
1: $R = \emptyset$; $indexList[] = \{\}$; $totalCost = 0$;
2: for all $c_i \in C$ do
3:     $I = createSFCIndex(c_i)$;
4:     $totalCost = \sum_{k=1}^{|W|} minCost(q_k, I)$;
5:     $indexList[i] = (I, totalCost)$;
6: end for
7: $sortAscending(indexList)$
8: $R = \bigcup_{j=0}^{o-1} IndexList[i].getIndex()$;
9: return $R \cup createSFCIndex(A)$;
```

The selection algorithm begins by creating an SFC-based index for each attribute combination in C and calculating the cost of the workload over the created indices (Algorithm 4: line 2–4). The indices along with their costs are stored in a list (Algorithm 4: line 5). The list is later sorted in an ascending order of the workload cost and the top $o-1$ indices are selected into R (Algorithm 4: line 7–8). Finally, an index created from the combination of all attributes is added to R, and R is returned as the set of recommended indices (Algorithm 4: line 9).

The core loop of the selection algorithm performs n_c executions, where n_c is the size of the candidate set C (Algorithm 4: line 2). Therefore, the complexity of the first complex statement of the algorithm (Algorithm 4: line 4) is calculated as $O\left(n_c \cdot |W|\right)$. The second complex statement of the algorithm is the call to the *sortAscending*() method (Algorithm 4: line 7). A good sorting algorithm, e.g., mergesort or heapsort, has a runtime complexity of $O\left(n_c \; log \; n_c\right)$ [CLRS03]. Hence, the overall worst-case complexity of the selection algorithm is given as:

$$O\left(n_c \cdot |W|\right) + O\left(n_c \; log \; n_c\right)$$

4.5 System Evaluation

In this section, we present results from the experimental evaluations of the index recommendation algorithms. The algorithms discussed above have been implemented in Java. An AMD Opteron machine with 4 GB of RAM, running Linux operating system has been used to perform these evaluations. The workloads for evaluating the index recommendation algorithms have been generated using the parameters defined in Table 4.2.

Parameter	**Definition**
m	Number of queries in the workload
n	Size of the attribute set A
r	Percentage of distinct attribute combinations in the workload
α	Zipfian distribution parameter

Table 4.2: Parameters for a Generating Workload

Since the index recommendation algorithms presented in this chapter are the first for P2P information discovery systems, evaluations using a variety of workload scenarios is required. Hence, a fine grained control over parameters such as total number of queries, query popularity distribution etc., is needed. Therefore, we use synthetic workloads for evaluating our algorithms. Moreover, unlike DBMS where benchmark workloads are made available by the TPC [tpc], no such workload of multi-attribute range queries is universally available for P2P systems. Workloads from P2P file sharing mostly contain multi-attribute point queries and are therefore not applicable here.

Using resource discovery in grid computing as a use-case, n number of attributes from the list shown in Table 4.3 are provided as an input to the workload generator. The workload generator creates a randomly ordered list of all attribute combinations from the provided attribute set. The list is then reduced by keeping only $r\%$ of the items and discarding the rest of them.

Attribute	**Value Domain**	**Definition**
CPU Speed	1.0 – 4.0	CPU clock speed in GHz
Busy CPU	0 – 100	Percentage of CPU(s) in use
Mem Size	1.0 – 8.0	Total Memory size in GB
Mem Used	0 – 100	Percentage of Memory in use
HDD Size	100.0 – 3000.0	Total HDD size in GB

Table 4.3: Attribute List for Generating Data and Workload

Next, each combination in the reduced list is assigned a popularity p using the Zipfian distribution with the parameter α. α is a decimal value between 0 and 1, where 0 represents uniform distribution (all combinations have the same popularity) and 1 represents highly skewed distribution (20% combinations make up 80% of all queries). The popularity of a combination indicates the number of times a combination is repeated in the workload queries. The sum of all popularities is equal to m (see Table 4.2).

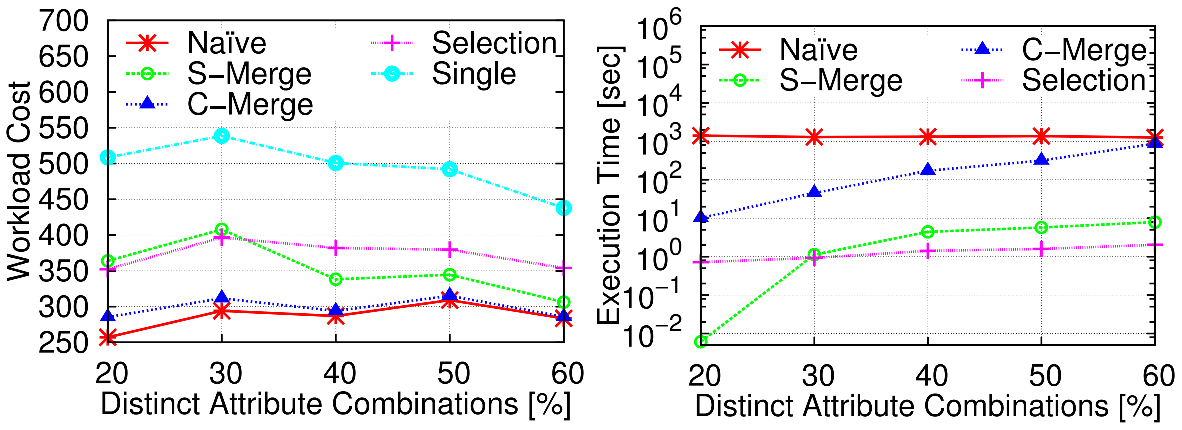
A multi-attribute range query is created by selecting an attribute combination from the list and randomly assigning a value range to each attribute of the selected combination. The assigned range for each attribute is chosen from the domain of the attribute shown in Table 4.3. Finally, a workload of queries is created by selecting each attribute combination p times for value range assignment, where p is the popularity of the attribute combination.

We use an attribute set of only 5 attributes as shown in Table 4.3. This is done in order to enable the comparison of the naïve index recommendation algorithm with the other algorithms, because the execution time of the naïve index recommendation algorithm becomes prohibitively high for a larger attribute set. At the same time, the naïve algorithm serves as a reference, since it leads to the optimal set of recommendations.

4.5.1 Performance Evaluation

In this section, we present results from the performance evaluation of all index recommendation algorithms. For the sake of comparison, the performance of a system where an index recommendation algorithm is not used, i.e., a system with only a single index created from the combination of all attributes, is also evaluated. For each of the evaluation scenarios discussed below, the following two performance metrics are measured:

- **Total Workload Cost** – Cost of all queries in the workload. The cost here refers to the estimated query cost discussed in Section 4.3, i.e., we do not consider the



(a) (b)

Figure 4.4: Influence of Varying Attribute Combinations

real cost of the workload in a DHT, e.g. the actual number of queried peers. An evaluation with a simulated DHT network follows later.

- **Execution Time** – Execution time of an algorithm in seconds.

For each point on the graphs displayed in this section, the corresponding experiment is repeated 10 times with different workloads, and an average value is plotted.

4.5.1.1 Influence of Varying Attribute Combinations

In this section, we study the effect of varying the number of attribute combinations in the workload. Table 4.4 shows the parameters used for performing this evaluation. The first four parameters of Table 4.4 are used by the workload generator to produce 5 different query workloads. Each new workload has a larger variety of attribute combinations then the previous one. The last parameter of Table 4.4 used by the index recommendation algorithms represents the user-defined limit for the maximum number of recommendations. Figure 4.4 shows the performance of each index recommendation algorithm.

Parameter	**Value(s)**
m	$10,000$
n	5
c	$20, 30, \ldots, 60$
α	0.8
o	3

Table 4.4: Evaluation Parameters – Influence of Varying Attribute Combinations

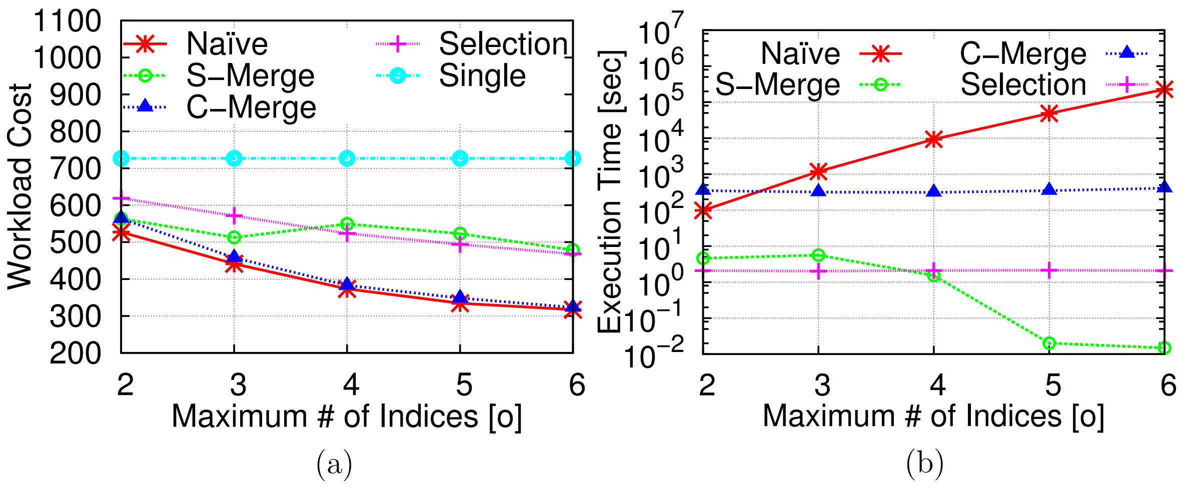


Figure 4.5: Influence of Varying Number of Indices

With respect to the total workload cost, the naïve algorithm performs best in all cases because the algorithm recommends a set of optimal indices for the workload (see Figure 4.4(a)). The scalable algorithm that comes closest to the optimal solution is the cost-based merge algorithm. The similarity-based merge algorithm performs slightly worse than the selection algorithm for the cases where Algorithm 2.2 (procedure for adding a missing index to the final set of indices) is mostly not executed (for, $r = 20$ & 30). In other cases, the selection algorithm shows worse performance than the similarity-based merge algorithm. Note that the total workload cost of all index recommendation algorithms is lower than the case where only a single index with the combination of all attributes is used.

Although the execution time of the naïve algorithm is highest compared to the execution times of the other algorithms (see Figure 4.4(b)), it remains almost constant because the search space of the naïve algorithm always includes all possible attribute combinations (see Section 4.4.1). Since the initial search space of the scalable algorithms only includes the attribute combinations from the workload (see Section 4.4.2), the execution time of the scalable algorithms grows with increasing number of unique attribute combinations in the workload.

Figure 4.4(b) also shows the limitations of the cost-based merge algorithm. If the number of attribute combinations is higher than 60%, the execution time of the cost-based merge algorithm exceeds the execution time of the naïve algorithm.

4.5.1.2 Influence of Varying Number of Indices

In this section, we demonstrate the effect of a varying user-defined limit for the maximum number of indices on each index recommendation algorithm. The parameters used for performing the evaluation are shown in Table 4.5.

Parameter	Value(s)
m	$10,000$
n	5
c	50
α	0.8
o	$2, 3, \ldots, 6$

Table 4.5: Evaluation Parameters – Influence of Varying Number of Indices

Generally, the total workload cost for each index recommendation algorithm decreases as the user-defined limit for the maximum number of indices increases (see Figure 4.5(a)). This happens because with increasing number of indices more queries are able to find less expensive indices for resolution.

The similarity-based merge algorithm produces higher workload cost than the selection algorithm for cases where Algorithm 2.2 is mostly not executed (Figure 4.5(a), for $o = 4, 5$ & 6). This indicates that the quality of indices produced by the similarity-based merge algorithm is better in cases where Algorithm 2.2 is executed, because Algorithm 2.2 selects the final index from the candidate set based on the total workload cost.

The total execution time of the naïve algorithm increases with increasing number of indices (see Figure 4.5(b)), because the complexity of the algorithm grows with increasing number of indices (see Section 4.4.1). However, the total execution time of the cost-based merge algorithm and the selection algorithm remains almost constant, because for these algorithms, the execution time is mostly dependent on the size of the candidate set which remains constant throughout the evaluation.

The total execution time of the similarity-based merge algorithm suddenly decreases between values 3 and 5 for the maximum number of indices (see Figure 4.5(b)). This happens because in these case Algorithm 2.2 is mostly not executed.

4.5.1.3 Influence of Varying Popularity Distribution

In this section, we illustrate the effect of varying query popularity distribution on each index recommendation algorithm. The parameters used for performing the evaluation are shown in Table 4.6.

We vary the query popularity distribution of the workload from uniform distribution ($\alpha = 0$) to highly skewed distribution ($\alpha = 1$). Figure 4.6 shows the performance of each index recommendation algorithm with respect to the total workload cost. The execution time of the algorithms is not shown because it remains almost constant throughout the evaluation, showing the same order as in the previous evaluation.

As expected, the naïve algorithm yields the least workload cost in all cases. The scalable algorithm that comes closest to the naïve approach is the cost-based merge algorithms.

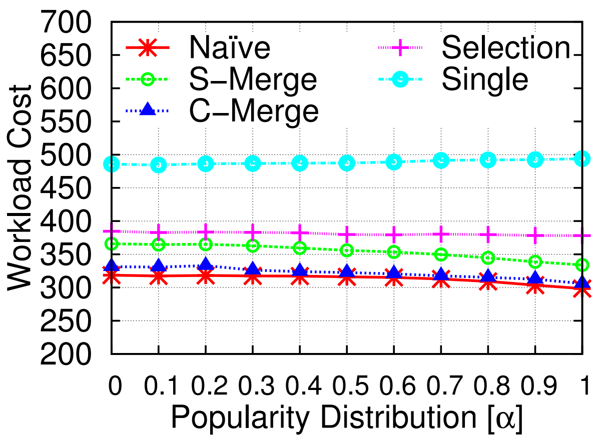


Figure 4.6: Influence of Varying Popularity Distribution

Parameter	**Value(s)**
m	$10,000$
n	5
c	50
α	$0.0, 0.1, \ldots, 1.0$
o	3

Table 4.6: Evaluation Parameters – Influence of Varying Popularity Distribution

The selection algorithm produces the highest workload cost which slightly decreases as the popularity distribution varies from uniform to skewed distribution. As with the previous evaluation, the total workload cost of all index recommendation algorithms is lower than the case where only a single index with the combination of all attributes is used.

Since all index recommendation algorithms generally try to include the most popular indices in the final set of recommendations, the total workload cost produced by each index recommendation algorithm decreases as the query popularity distribution varies from uniform to skewed distribution.

4.5.2 Network Simulation

In this section, we show that the SFC-based cost estimation formula presented in Section 4.3 is accurate enough for a qualitative comparison between different index recommendation algorithms discussed in Section 4.4. In order to do so, we first evaluate each index recommendation algorithm using the parameter values shown in Table 4.7. Each evaluation experiment is repeated 5 times with a different workload. The estimated

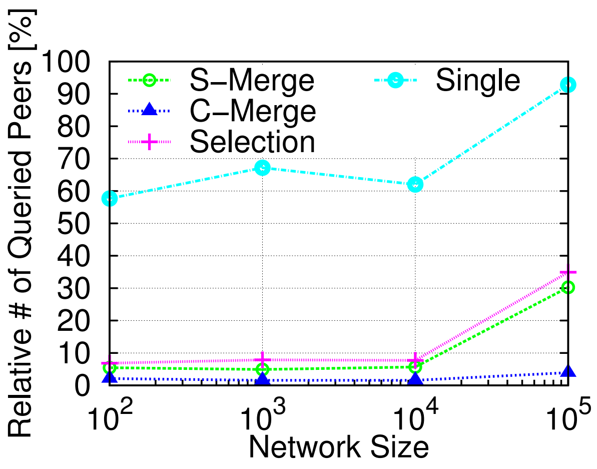


Figure 4.7: Network Simulation

total workload cost, averaged over 5 runs, calculated by each index recommendation algorithm, is shown in the following Table 4.8. This cost is the estimated cost of the workload calculated using Equation 4.2 discussed in Section 4.3.

Parameter	**Value(s)**
m	50
n	4
c	50
α	0.6
o	3

Table 4.7: Evaluation Parameters – Network Simulation

The OID System [MTD$^+$08] is then set up in a P2P network simulation environment. Multiple SFC-based indices, corresponding to the recommendation given by an index recommendation algorithm, are defined in the OID Index Space (see Section 4.2). Each query from the same workload used above is then issued from a random peer in the network. This experiment is repeated 5 times for each index recommendation algorithm using the 5 corresponding workloads utilized during the evaluation above. The following metric is then measured and averaged over 5 runs for each recommendation algorithm:

- **Number of Queried Peers** – Total number of peers queried in order to resolve all the queries in the workload.

Table 4.8 shows that the naïve index recommendation algorithm performs best with respect to the total workload cost, followed by the cost-based merge, similarity-based merge and the selection algorithm respectively. The total workload cost is highest for a single index created using the combination of all attributes.

Algorithm	**Total Workload Cost**
Naïve	1.739
S-Merge	2.038
C-Merge	1.776
Selection	2.084
Single	2.852

Table 4.8: Total Workload Cost

Figure 4.7 shows the number of queried peers for each scalable index recommendation algorithm (in percentage) relative to the number of queried peers for the naïve index recommendation algorithm. The relative number of queried peers are also shown for a single index created using the combination of all attributes. Figure 4.7 asserts the same order of the algorithms as seen in Table 4.8, but now with respect to the actual number of queried peers. This shows that the cost estimation formula shown in Equation 4.2 is accurate enough for a qualitative comparison between different index recommendation algorithms.

Figure 4.7 also shows that in the best case, the cost-based merge algorithm queries only 1.5% more peers compared to the naïve index recommendation algorithm during the evaluation.

4.5.3 Evaluation Summary

The evaluation results discussed in Section 4.5.1 and Section 4.5.2 show that there is a trade-off between the performance and the execution time of the index recommendation algorithms. The algorithms that produce efficient index recommendations generally take longer time to do so compared to the algorithms that produce less efficient recommendations.

Among the scalable index recommendation algorithms, the cost-based merge algorithm performs the best with respect to the total workload cost (see Section 4.5.1) and the number of queried peers (see Section 4.5.2). The cost-based merge algorithm is generally followed by the similarity-based merge algorithm and the selection algorithm respectively.

With respect to the execution time, the cost-based merge algorithm shows the worst performance, generally followed by the similarity-based merge algorithm and the selection algorithm respectively. Although during some evaluations the execution time of the selection algorithm exceeds the execution time of the similarity-based merge algorithm (Figure 4.4(b) & Figure 4.5(b)), the worst-case analysis of the two algorithms (see Section 4.4.2.2 & Section 4.4.2.3) suggests that for a large variety of queried attribute combinations, the execution time of the similarity-based merge algorithm would be higher than the execution time of the selection algorithm.

4.6 Related Work

To the best of our knowledge, there are no index recommendation algorithms available for DHT-based information discovery systems. Nonetheless, index tuning/recommendation techniques have been widely studied in the area of database management systems (DBMS).

There is a fundamental difference between the index recommendation algorithms for databases and the algorithms presented in this chapter. The index recommendation algorithms for databases rely on the *SQL Query Optimizer* for evaluating the cost of a query over an index [BC05, CN97, CN99, VZZ+00, ZRL+04]. The estimated cost of a query given by the *SQL Query Optimizer* is highly accurate, because the query optimizer uses a central repository known as the *Data Dictionary*, containing statistical information about the data. Gathering such statistical information in a large, distributed and dynamic P2P network would result in a high overhead. Therefore, our approach only uses the local structural information of the indices to estimate the query cost (see Section 4.3).

The index recommendation algorithms designed by Bruno et al. [BC05], Chaudhuri et al. [CN97,CN99], and Valentin et al. [VZZ+00] carry out the recommendation process in two steps. In the first step, the execution of the workload queries is simulated. During the simulation process, the indices that appear in the *Query Execution Plans* of the query optimizer are collected. This collection represents the initial set of useful indices. The scalable index recommendation algorithms discussed in this chapter also provide the recommendations in two steps. However, our algorithms obtain the initial set of indices using the unique attribute combinations that appear in the workload queries (see Section 4.4.2).

During the second step of the index recommendation process, the recommendation algorithms for databases refine the initial set of indices using certain heuristics. The goal in general is to obtain a set of indices that produces the least cost for the queries in the workload.

Given a user-defined limit o for the maximum number of indices, the heuristic presented in [CN97] generates all possible m-sized ($m < o$) subsets of the initial set of indices. The subset with the least cost for the workload is then chosen as a seed for further processing. Next, an index from the initial set is continuously added to the seed until the size of seed is equal to o. It is unclear, how the value for m is chosen, since the algorithm has prohibitively high execution time if m is large, and sub-optimal recommendations if m is small.

In [CN99], the heuristic introduced by Chaudhuri et al. reduces the size of the initial set of indices by merging pairs of indices as long as a cost constraint is not violated. Unlike the cost-based merge algorithm (see Section 4.4.2.1) presented in this chapter that tries to minimize the workload cost given a limit o for the maximum number of indices, the algorithm in [CN99] minimizes the storage, given a cost constraint. Therefore, it is possible that the final set of recommendations is larger than o.

The algorithm introduced by Bruno et al. in [BC05] produces a set of recommendations by applying different transformations (merging, prefixing, deletion) on pairs of indices in the initial set. The algorithm continues as long as a certain time constraint is not violated. It is not clear, how long the algorithm should be executed so that it produces an o-sized set of recommendations.

Valentine et al. introduced a heuristic in [VZZ+00] based on a variation of the solution to the Knapsack problem. Their algorithm assigns a cost-to-benefit ratio to each index in the initial set of indices and then selects the indices with the highest ratio as long as a storage constraint is not violated. The algorithm then randomly swaps some indices from the set of selected indices with some indices in the initial set to try another variation of the solution. The swap and selection process continues as long as a time constraint is not violated. Although, the swap and selection technique could lead to a better solution than the initial one, it could also lead to a worse solution.

4.7 Conclusion

In this chapter we presented several index recommendation algorithms for DHT-based information discovery systems. Given a limit for the maximum number of indices and a workload of queries, each algorithm recommends a set of indices that produces close-to-optimal performance for the workload queries within the given limit. The set of index recommendation algorithms includes three scalable index recommendation algorithms: cost-based merge, similarity-based merge and selection algorithm.

Our evaluations show that there is a trade-off between the performance and the execution time of the scalable index recommendation algorithms. With respect to the performance, the cost-based merge algorithm is the best (only 1.5% worse than the naïve algorithm), generally followed by the similarity-based merge and the selection algorithms. With respect to the execution time of the algorithms, the order is reversed.

The index recommendation algorithms discussed in this chapter provide an aid to the designer of a distributed application in order to define useful indices for the expected queries in the system. The next chapter discusses the automation of index recommendation and installation in DHT-based information discovery systems. For that, we need to consider the overhead of installing an index compared to the benefit of installing it. Moreover, development of a distributed query monitoring service for gathering the workload of queries in DHTs is also discussed in the next chapter.

Chapter 5

Self-adapting Optimized Information Discovery System

5.1 Introduction

In the previous chapter, we presented a set of index recommendation algorithms that assists the designer of a distributed application to define a useful set of indices for efficiently supporting multi-attribute range queries over distributed hash tables (DHTs). Given a limit for the maximum number of indices and a representative set of multi-attribute range queries (*workload*), each index recommendation algorithm recommends a set of indices that produces close-to-optimal system performance for the workload within the given limit. The index recommendation algorithms work on the assumption that a workload of queries in a DHT-based information discovery system remains stable for a certain period of time, i.e., newly issued queries in the system are not significantly different from the previously issued ones. Based on this assumption, if an optimal set of indices is defined for the previously issued queries, then this set of indices should also remain close to optimal for the future queries issued over the same period of time.

The index recommendation algorithms work offline, i.e., it is assumed that the workload provided to an algorithm is somehow collected from an already existing DHT-based information discovery system. Further, it is assumed that the recommended set of indices is manually installed over the DHT by the designer of the distributed application. In order to carry out such an installation, the information discovery system would have to be taken offline, which might be feasible for small peer-to-peer (P2P) applications but highly undesirable for large-scale applications. In this chapter, we relax these assumptions to present an adaptive OID system. The adaptive OID system performs the task of index recommendation and index installation online, effectively eliminating the need for manually updating the indices.

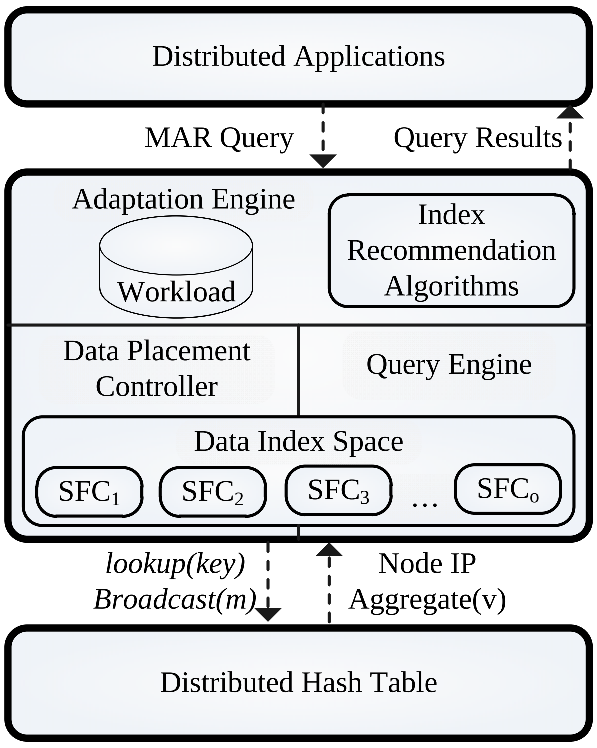


Figure 5.1: Architecture of the Adaptive OID

The main discussion in this chapter is regarding an index adaptation process. This index adaptation process, including online index recommendation and index installation, is carried out in four phases in a DHT. During the first phase, a workload of multi-attribute range queries is collected from several peers in the network using uniform random sampling. The second phase involves execution of an index recommendation algorithm to determine an optimal set of indices for the collected workload. During the third phase, the cost and the benefit of installing the recommended set of indices is calculated. If it is beneficial to install the recommended set of indices, the installation is carried out during the fourth phase.

The rest of the chapter is organized as follows: the architecture of the adaptive OID system is discussed in Section 5.2. In Section 5.3 we describe the index adaptation process and its phases in detail. Evaluation results are presented in Section 5.4. In Section 5.5 we give an overview of the related work. Finally, we conclude this chapter in Section 5.6.

5.2 Evolution of the OID Architecture

The basic OID architecture presented in Section 3.2 has evolved into the architecture of the adaptive OID system shown in Figure 5.1. There have been two fundamental advances. First, the OID framework layer (middle layer) has a new component known as the adaptation engine. The adaptation engine is responsible for online recommendation and installation of indices in the data index space of the OID framework layer. Second, in addition to the basic lookup functionality, we assume a DHT layer (bottom layer) that also provides the services for broadcasting a message and aggregating a value. The top layer (distributed application layer) of the architecture and other components of the OID framework layer stay the same as presented earlier.

5.3 Index Adaptation

The goal of the index adaptation process is to update the set of indices in the OID framework layer of each peer according to the dynamic workload of multi-attribute range queries in the system. In order to achieve this goal, we introduce a four-phase index adaptation process that is periodically executed in the system. The four phases of the index adaptation process are: distributed workload collection, index recommendation, adaptation decision, and index installation. During the first phase, a workload of multi-attribute range queries is collected from random peers in the network. This workload is used in the second phase to obtain a new set of indices. During the third phase, the cost and the benefit of installing the new set of indices is calculated. Finally, if it is beneficial to install the new set of indices, the installation is carried out during the fourth phase.

We define following three types of peer roles to carry out the index adaptation process:

- **Adaptation Peer** – An adaptation peer is a peer that periodically initiates the index adaptation process. The length of a period is set by the designer of the distributed application. In order to avoid conflicting index updates, there can only be a single adaptation peer at a time in the network. We assume that the location of the adaptation peer is pre-selected by the designer of the distributed application. This could be done by deciding that the peer that is the successors of a certain key in the DHT would be the adaptation peer in the network.

  If a new peer joins at the location of the adaptation peer, the state of the adaptation peer is transferred to it and this peer becomes the new adaptation peer. Moreover, if the adaptation peer fails during the first three phases of the index adaptation process, the process is restarted by the new adaptation peer. We assume a correctly functioning DHT where any peer that fails or leaves the network is automatically replaced.

- **Monitoring Peer** – Each peer in our system is a monitoring peer. Monitoring peers are involved in the local collection of the query workload, i.e., each monitoring peer logs each query that it resolves. This log is emptied when a new set of indices is installed in the data index space of the peer.

- **Sampling Peer** – A sampling peer in our system is a peer that is involved in distributed workload collection discussed in the next section. Any peer can take the role of a sampling peer.

5.3.1 Distributed Workload Collection

Ideally, if the complete set of past queries were collected from all peers in the network, the optimal set of indices could be obtained during the second phase of index adaptation. However, collecting queries from all peers is neither efficient nor scalable. Therefore, the goal of the distributed workload collection phase is to collect a subset of the complete set of queries by sampling some random peers. The idea is to sample a sufficiently large subset of peers at different locations in the network to get an approximation of the complete set of queries.

The adaptation peer could directly collect a workload of multi-attribute range queries by randomly sampling some monitoring peers in the network. However, in that case, the adaptation peer will have to issue a large number of sampling requests and handle a large number of sampling responses, making the sampling process unscalable. Therefore, in order to limit the fanout of the adaptation peer and to make the sampling process scalable, we use a two-level sampling process. The two-level sampling algorithm presented below could be easily extended to n levels. However, the larger the number of levels, the higher the network load for the sampling process will be.

The adaptation peer initiates the first level of the sampling process by generating β random keys from the identifier space of the DHT, i.e., $[0, 2^m)$, where m is the number of identifier bits. A DHT lookup is then performed for each random key in order to identify the peer responsible for it. Here, we assume a basic DHT lookup functionality that, given a key, returns the identity of the peer responsible for the key. Once the identity of a random peers is learned, a sampling request with parameter γ is sent to it, where γ indicates the number of peers to be sampled at the second level of the sampling process.

Upon receiving a sampling request from the adaptation peer, a peer assumes the role of a sampling peer. The sampling peer then forwards the sampling request to γ random monitoring peers in the same manner as the adaptation peer. After receiving a sampling request from a sampling peer, a monitoring peer sends the local query workload to the sampling peer.

A sampling peer accumulates all the workloads received from γ random monitoring peers into a single workload. Since the same query could have been resolved by several monitoring peers, it could appear multiple times in the accumulated workload. Therefore, duplicates are eliminated during the accumulation process. Note that the same query issued twice is not considered as a duplicate query since each query has a globally unique identifier. A globally unique identifier for a query is generated by

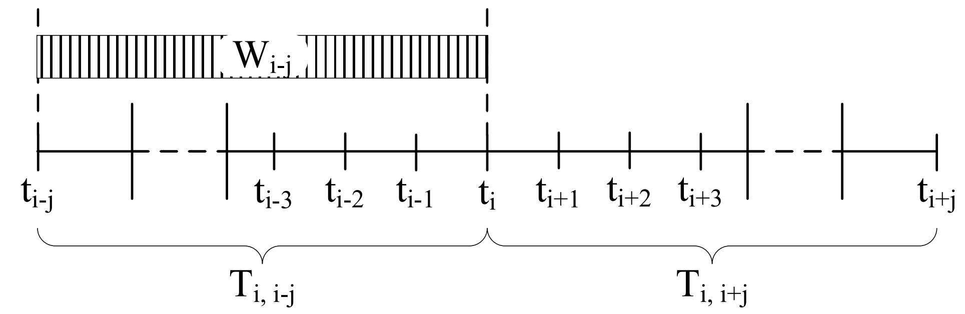


Figure 5.2: Adaptation Decision

concatenating the unique identifier of the querying peer with a numeric counter. Finally, the accumulated workload including the workload of the sampling peer is sent to the adaptation peer where the accumulation process is repeated.

In order to detect the failures of the monitoring or the sampling peers, the process of distributed workload collection includes timeouts at each level. At the level of a sampling peer, if a response is not received from a monitoring peer before the timeout, the sampling request is re-issued assuming that the faulty monitoring peer has been replaced by the DHT. Similarly, at the level of the adaptation peer, if a response is not received from a sampling peer before the timeout, the sampling request is re-issued.

In case of the Chord DHT (see Section 2.2.2.1), the distributed workload collection phase requires $O\left((\beta \cdot \gamma) \cdot (log_2 N + 2)\right)$ messages in the worst-case to collect a workload of multi-attribute range queries. N is the total number of peers in the network and $log_2 N$ is the maximum number of messages required for a lookup. Two additional messages are needed to send a sampling request to a peer and receive a sampling response from it.

5.3.2 Index Recommendation

Once a workload of multi-attribute range queries has been collected at the adaptation peer, the next step in the adaptation process is to search for an optimal set of indices for the collected workload. For this purpose, we utilized the index recommendation algorithms introduced in the previous chapter.

Given a workload of multi-attribute range queries and a limit o for the maximum number of indices, an index recommendation algorithm recommends a close-to-optimal set of indices I_r for the given workload. For a detailed description of the index recommendation algorithms, see Chapter 4.

5.3.3 Adaptation Decision

After obtaining a recommended set of indices from an index recommendation algorithm, a naïve approach would be to directly install this set of indices in the network. However, it is possible that the cost of installing the recommended set of indices outweighs the benefit of installing it. Therefore, the goal of the adaptation decision phase is to determine whether the installation of the recommended set of indices is beneficial or not. This is done by comparing the estimated cost of the workload over the current set of indices with the estimated cost of the workload over the recommended set of indices. The installation cost of the recommended set of indices is also taken into account.

Let t_i mark the current periodic execution of the index adaptation process, I_c be the current set of indices, and I_r be the recommended set of indices. Then, we define the following quantities in our system (see Figure 5.2):

- $\mathbf{T_{i,i-j}}$ – Time interval between t_i and t_{i-j} $\forall j \in \mathbb{N}^+$ where, t_{i-j} marks the index adaptation process where I_c was installed. Note that this time interval is dynamic since a new set of indices is not installed during each periodic execution of the index adaptation process.
- $\mathbf{W_{i-j}}$ – Complete set of multi-attribute range queries during the time interval $T_{i,i-j}$.
- $\mathbf{SW_{i-j}}$ – Sampled workload, from the complete set of multi-attribute range queries during the time interval $T_{i,i-j}$.
- $\mathbf{cost_{in}}$ – Estimated cost of installing the recommended set of indices I_r.

The adaptation peer considers the installation of the recommended set I_r beneficial, if the following condition holds:

$$cost(SW_{i-j}, I_c) > cost(SW_{i-j}, I_r) + cost_{in} \tag{5.1}$$

i.e., if the cost of the sampled workload SW_{i-j} over the current set of indices I_c is greater than the cost of the same workload over the recommended set of indices I_r plus the installation cost of the recommended set of indices. The assumption behind Condition 5.1 is that the complete set of multi-attribute range queries W_{i-j} would be repeated for a similar interval of time in the future, i.e., for $T_{i,i+j}$. This is the most general assumption for predicting the cost of future queries. If Condition 5.1 is satisfied, the next phase of index adaptation is carried out. Otherwise, the index adaptation process is halted until the next periodic execution.

The cost functions $cost(SW_{i-j}, I_c)$ and $cost(SW_{i-j}, I_r)$ in Condition 5.1 can be generalized as a cost function $cost(Q, I)$. If $Q = \{q_1, q_2, q_3, \ldots, q_l\}$ is a set of queries and $I = \{SFC_1, SFC_2, SFC_3, \ldots, SFC_o\}$ is a set of indices, then $cost(Q, I)$ is calculated as:

$$\begin{aligned} &cost(Q,I) = \sum_{i=1}^{|Q|} cost(q_i, SFC_j), \text{ such that} \\ &cost(q_i, SFC_j) < cost(q_i, SFC_k), \text{ where} \\ &\forall j,k : 1 \leq (j,k) \leq |I| \text{ and } j \neq k \end{aligned} \tag{5.2}$$

i.e., the cost of a set of queries Q over a set of indices I is a sum of the cost of each query in Q over the least expensive index in I.

In order to determine the least expensive index for a query, the network cost of the query over each index needs to be calculated. Due to highly dynamic nature of P2P systems, this cost cannot be accurately anticipated. However, if the cost of routing a message in the network is known, the maximum cost for resolving a query can be calculated.

Let z be the total number of zones a query maps to, on an SFC-based index. In order to resolve this query, the peer responsible for each zone has to be queried. If a basic query routing strategy is considered, where first a lookup is performed to determine the peer responsible for each zone, then the maximum cost of a query q on an index SFC is calculated as:

$$cost(q, SFC) = z \cdot (log_2 N + 2) \text{ [messages]} \tag{5.3}$$

where N is the total number of peers in the network and $log_2 N$ is the maximum number of messages needed for looking up a peer responsible for a zone. Two additional messages are needed to send a query request to a peer and receive a query response from it.

This formula for estimating the cost of a query on an SFC-based index is more accurate than the cost estimation formulas presented in Section 3.6 and Section 4.3. The reason is that this formula relies on actual number of peers (N) in the network to estimate the message load produced by a query. Although this formula is dependent on the type of used DHT (Chord is assumed here) and the used query routing algorithm (the most basic one is assumed here), the index adaptation process remains independent of the formula. Any other formula that allows qualitative comparison of SFC-based indices could be used here.

In order to check if Condition 5.1 holds, the cost of installing the recommended set of indices $cost_{in}$ has to be calculated. Similar to the cost calculation above, only the maximum cost of installation can be calculated. Let λ be the total number of unique data objects in the system, then the maximum cost for installing the recommended set of indices I_r is calculated as:

$$\begin{aligned} cost_{in} = 3 \cdot (N-1) + (|I_r| - |I_c \cap I_r|) \cdot \\ \lambda \cdot (log_2 N + 2) \text{ [messages]} \end{aligned} \tag{5.4}$$

where $3 \cdot (N-1)$ is the cost of broadcasting the recommended set of indices, $(|I_r| - |I_c \cap I_r|)$ is the total number of new indices, and $\lambda \cdot (log_2 N + 2)$ is the cost of re-indexing the data. The reason for the broadcast cost being almost three-times the network size is discussed in the next section.

This formula for estimating the cost of re-indexing the data objects is also DHT dependent (Chord is assumed here). However, it could be replaced by any other formula that accurately estimates the cost of re-indexing the data objects in any DHT.

Note that Equations 5.3 and 5.4 require the knowledge of global parameters like N and λ, which are generally not known to a peer in a DHT network. However, an estimate for these parameters could be obtained by installing a reliable broadcast/aggregation tree in the network. The root of this tree will be the adaptation peer in our system. Such a broadcast/aggregation tree could be installed and maintained using the approaches discussed in [EAABH03] and [HZ10].

5.3.4 Index Installation

Once it is determined that installing a recommended set of indices is beneficial, the adaptation peer initiates the index installation phase. The goal of the index installation phase is to broadcast the new set of indices I_r, and to re-index the data on each peer accordingly. A data object is re-indexed by removing the identifier assigned to it using an old index (an index in I_c but not in $\{I_c \cap I_r\}$) and then assigning it a new identifier using a new index (an index in I_r). When the new identifier is assigned to the data object, it is possible that this identifier does not fall under the identifier space of the current peer. In that case, the data object is relocated to the peer that is now responsible for hosting it.

A naïve way of carrying out the index installation phase is to broadcast the recommended set of indices using the DHT broadcast/aggregation tree, and let each peer re-index the data according to the new set of indices. However, this way the queries issued in the system during the re-indexing process may not be able to find all the matching data objects. This could happen in cases where, e.g., a query issued using a new index, searches for matching data objects at a peer where the data has not been placed using this index yet. In order to avoid such situation, we introduce a 3-step index installation phase.

During the first step, a broadcast message containing the new set of indices I_r is sent by the adaptation peer to each peer in the system. For this purpose, the DHT broadcast/aggregation tree is used. Upon receiving the broadcast message, each peer begins the data re-indexing process discussed below. Note that the queries that are issued during this step continue to be resolved using I_c.

As discussed earlier, the data index space of the OID system consists of o number of indices, i.e., $|I_c| = o$. A data object is indexed using each of these indices. This means that there are o copies (references) of the same data object with different identifier in the system (possibly residing at o different peers). Now if each copy of the data object

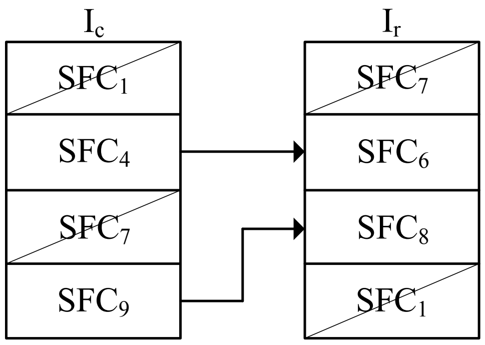


Figure 5.3: Data Re-Indexing

is re-indexed using each new index in I_r, the copies of the data object would increase by $o - 1$ times $|I_r|$. For example, if I_c and I_r are as shown in Figure 5.3, a data object indexed using I_c would be located at four locations in the system since $|I_c| = 4$. Now if each of those four locations re-indexes the data object using each index in I_r, there would be 16 copies it in the system as (o x $|I_r|$) = (4 x 4) = 16. To avoid this situation, it has to be made sure that each copy of the data object is not re-indexed using each new index. This is achieved as follows.

Data re-indexing at a peer starts with the comparison of the installed set of indices I_c with the new set of indices I_r (see Figure 5.3). First, the common elements in both sets are ignored because no re-indexing is needed using indices that already exist and are also a part of I_r. Next, a mapping is defined from each element in I_c to each corresponding element in I_r. A data object that had been previously indexed using an element of I_c is now re-indexed only using the corresponding element in I_r. The re-indexing process involves a lookup for the peer responsible for the new identifier of the data object and the relocation of the data to that peer.

Consider the set of indices shown in Figure 5.3. Using the re-indexing technique discussed above, the peer that held the copy of a data object with an identifier assigned using SFC_4 in I_c, will re-index this copy using SFC_6 in I_r. Similarly the peer that held the copy of the data object with an identifier assigned using SFC_9 in I_c, will re-index this copy using SFC_8 in I_r. Since each copy of a data object is re-indexed only once, the number of copies of the data object remains constant in the system. Note that for this process to work, each copy of the data object must keep track of the index used for assigning an identifier to it.

Once the re-indexing of the data is finished at a peer, it sends an acknowledgement message to the parent node in the DHT broadcast/aggregation tree. During the second step of the index installation process, the acknowledgements from all peers in the network are aggregated until the adaptation peer receives the aggregated acknowledgement. Queries still continue to be resolved using the old set of indices I_c.

Upon receiving an aggregated acknowledgement from the child nodes in the DHT broadcast/aggregation tree, the adaptation peer starts the third step of index installation by broadcasting a `use index` message. When this message is received at a peer, the peer removes I_c, discards the corresponding data, empties the monitored query log, and starts using I_r for query resolution. Note that the data common between I_c and I_r is not discarded. During this step of index installation, if a query is issued from a peer that has not received the `use index` message yet, then there are two possibilities. First, the query will be resolved using I_c, if all peers involved in query resolution have not discarded the data corresponding to I_c. Second, even if a single peer involved in query resolution has discarded I_c, then the peer that issued the query will be asked to re-issue it using I_r.

If the adaptation peer fails before the first step of the index installation phase, then the process of index adaptation is repeated by the new adaptation peer. However, if the adaptation peer fails after the first step of index installation, the new adaptation peer is already aware of the state of index installation due to the broadcast of new indices in the network. Therefore, the new adaptation peer executes the next steps of the index installation phase.

5.4 System Evaluation

In this section, we present the results from the performance evaluation of the adaptive OID system. We simulated our system using the PeerSim [pee] P2P overlay simulator. The simulations were performed on an AMD Opteron machine with 4 GB of RAM.

Attribute	Value Domain	Definition
CPU Speed	1.0 – 4.0	CPU clock speed in gigahertz
Busy CPU	0 – 100	Percentage of CPU(s) in use
Mem Size	1.0 – 8.0	Total Memory size in gigabytes
Mem Used	0 – 100	Percentage of Memory in use
HDD Size	100.0 – 3000.0	Total HDD size in gigabytes
DL Bandwidth	0.5 – 100	Bandwidth of down link in mbits/sec

Table 5.1: Attribute List for Generating Data and Query Workloads

Considering resource discovery in grid computing as an example scenario, we represent the data objects in our simulations as resource specifications. Each resource specification consists of attributes shown in Table 5.1. The value for each attribute in a resource specification is randomly generated from the value domain of the attribute. A typical generated resource specification looks like: (`CPU Speed` = 1.2 GHz, `Busy CPU` = 60%, `Mem Size` = 1024 MB, `Mem Used` = 289 MB, `HDD Size` = 50 GB, `DL Bandwidth` = 16 mbits/sec). Note that the generated resource specifications do not influence the evaluation results, as they are based on the performance of multi-attribute range queries.

The performance of multi-attribute range queries depends upon the indices used for indexing the data and not on the values used for attributes in data objects.

Unlike the database management systems where benchmark workloads are made available by the TPC [tpc], no such workload of multi-attribute range queries is universally available for P2P systems. Therefore, we generate the workloads using the attributes in Table 5.1 for simulating different scenarios of our system. A typical example of a generated query looks like: (`CPU Speed` $>$ 1.0 GHz)$\wedge$(`CPU Speed` $<$ 3.0 GHz)$\wedge$(`Mem Size` $>$ 512 MB).

For each point on the graphs displayed in this section, the corresponding experiment is repeated 10 times with different workloads, and an average value is plotted. Moreover, the index recommendation algorithm used throughout the evaluations is the cost-based merge algorithm (see Section 4.4.2.1).

5.4.1 Varying Number of Attributes

In this section, we present the results from the performance evaluation of our system using a workload of queries with varying number of attributes. We show that an adaptive OID system is essential for continuous optimization of overall system performance for multi-attribute range queries. Table 5.2 shows the parameter values used for this simulation.

Parameter	**Value**	**Definition**
N	1000	Total number of peers in the DHT
n	1600	Total number of queries in the workload
o	3	Maximum number of indices
λ	5000	Total number of data objects
β	33	First level sampling parameter
γ	2	Second level sampling parameter

Table 5.2: Evaluation Parameters – Varying Number of Attributes

The queries in the workload are generated so that the start of the workload contains queries with 4 attributes followed by queries with 3, 2, and 4 attributes again. To simulate a slow change in the workload over time, the attributes in the queries are varied slowly. First, only the queries with 4 attributes are included in the workload followed by a mixture of queries with 4 and 3 attributes. As the generation process continues, the frequency of queries with 4 attributes keeps reducing until only queries with 3 attributes are in the workload. Then, a mixture of queries with 3 and 2 attributes is included in the workload, and so on. It is important to simulate a slowly changing workload to analyse the system's adaptation capabilities to a slow change. If the system reacts to a slowly changing trend, it is imminent that a faster changing workload would also trigger a reaction.

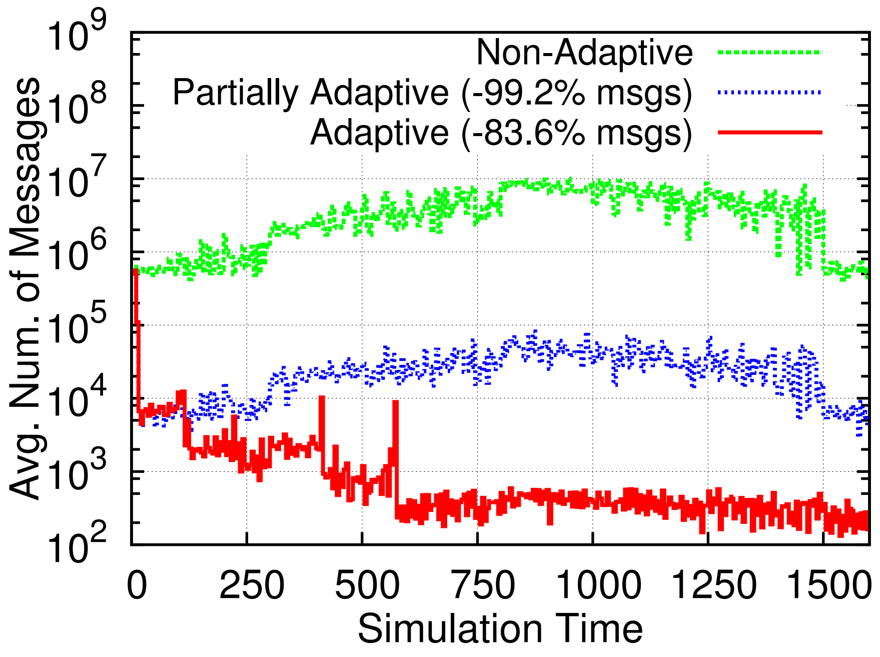


Figure 5.4: Varying Number of Attributes

Each attribute in a query is randomly selected from the list shown in Table 5.1. Similarly, the range for an attribute in a query is randomly selected from the domain of the attribute. The values for parameters β and γ are set so that almost 10% of peers in the network are sampled.

We simulate the adaptive OID system, the non-adaptive system, and a system with only a single adaptation (partially adaptive system), by executing the generated workload from random peers in the DHT over a period of time. The non-adaptive system is a system with only a single data index over all 6 attributes shown in Table 5.1. For the partially adaptive system, the adaptation takes place after 10 simulation time units. Moreover, for the adaptive OID system, the index adaptation process is scheduled to run after every 10 simulation time units. A single simulation time unit is long enough to allow execution of a single query.

For every 5 simulation time units, we plot the average number of messages in the system during that 5-time-unit-window (see Figure 5.4). The number of messages represents all the messages in the system including messages for the index adaptation process. This metric is plotted for all three systems. For the adaptive OID system, the peaks in Figure 5.4 mark the points where index installation takes place. The higher the peak, the larger the number of indices that are exchanged. Some small peaks are levelled out due to averaging over 10 simulation runs.

Similar to the non-adaptive system, the adaptive OID system and the partially adaptive system start with one index over all attributes. However, the first adaptation happens

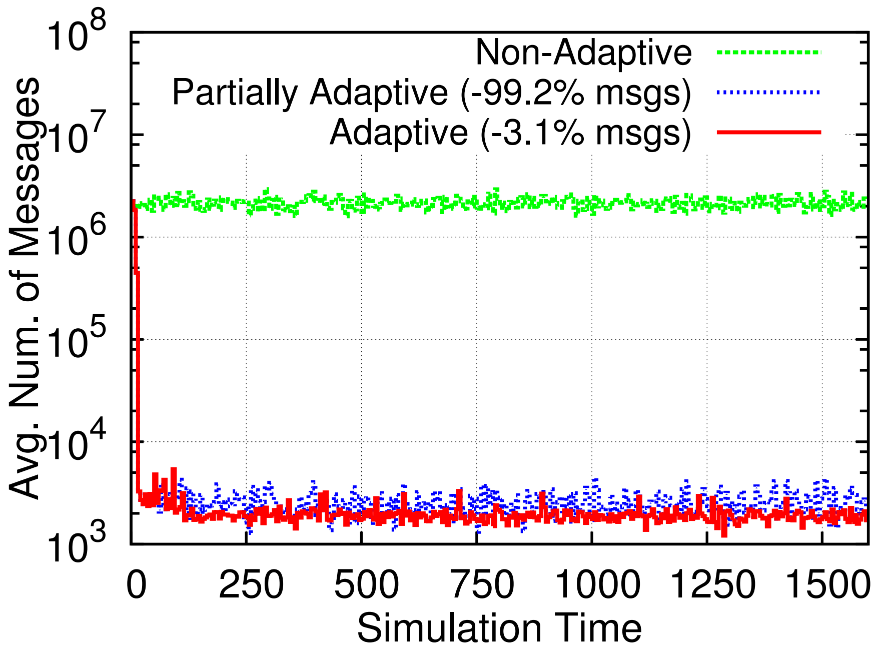


Figure 5.5: Fixed Number of Attributes

very soon in both the systems and 2 additional indices are installed (see Figure 5.4). This improves the performance of multi-attribute range queries in both systems because the queries are able to find less expensive indices for resolution. Since the first adaptation is based on a very small workload, the second adaptation follows soon in the adaptive OID system. The system continues to adapt itself over time according to the workload of queries. After each adaptation, the performance of multi-attribute range queries improves as the average number of messages in the system are reduced.

Figure 5.4 shows that the partially adaptive system has 99.2% less messages compared to the non-adaptive system. Moreover, the adaptive OID system has 83.6% less messages compared to the partially adaptive system. Therefore, the adaptive OID system is several orders of magnitude better than the non-adaptive system. Figure 5.4 also shows that, in order to optimize the overall system performance for multi-attribute range queries, a system with continuous adaptations is essential.

The performance of the non-adaptive system worsens with decreasing number of attributes in queries (see Figure 5.4). This happens because with decreasing number of query attributes, more attributes have to be considered as wild-cards on a single large index. The performance of the system gets better towards the end of the simulation because the number of attributes in queries increases from 2 to 4 attributes.

In order to further analyse the impact of the number of attributes in queries, we perform another simulation where the number of attributes in the workload is kept constant to 3 attributes. Other simulation parameters have the same values as in Table 5.2. Figure

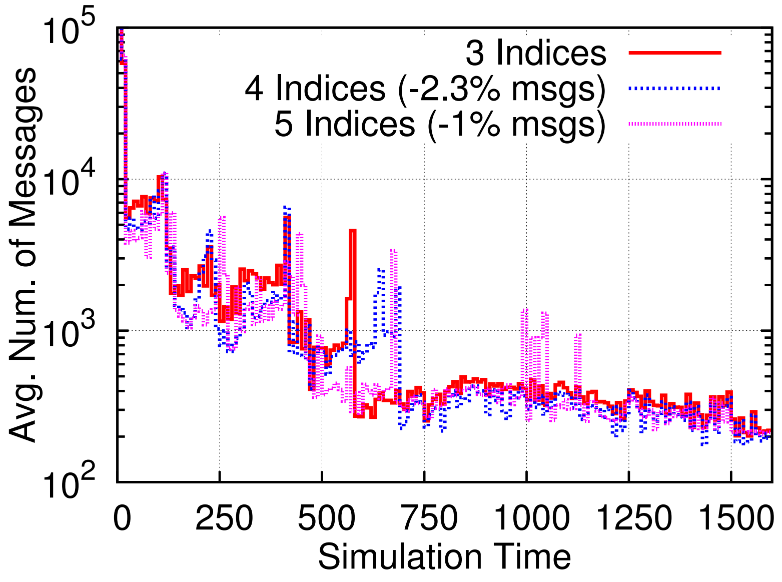


Figure 5.6: Varying Number of Indices

5.5 shows the performance of all three systems with respect to the average number of message in a 5-time-unit-window.

Figure 5.5 shows that the adaptive OID system quickly adapts its indices to the changing workload of queries. Major adaptations come close to the start of the simulation. After that, even though some small adaptations happen in the system, the performance of the system remains roughly constant. This happens because the indices adapted during the start of the simulation remain beneficial for the complete simulation. The performance of the non-adaptive system remains almost constant, and several orders of magnitude worse than the adaptive system, throughout the simulation.

With a constant number of attributes in queries, the performance of the partially adaptive system comes quite close to the performance of the adaptive system (see Figure 5.5). However, the adaptive system still produces 3.1% less messages compared to the partially adaptive system. This difference in the number of messages grows larger over time. Therefore, in a long running system, the adaptive system would perform significantly better than a partially adaptive system.

5.4.2 Varying Number of Indices

In this section, we present the performance evaluation of the adaptive OID system by showing the impact of varying number of indices on the system. We perform 3 different simulations using the same workload as in the first simulation discussed in Section 5.4.1. The parameters used for performing these simulations are shown in Table 5.3. For each simulation we plot the average number of messages in a 10-time-unit-window.

Parameter	Value
N	1000
n	1600
o	3, 4, 5
λ	5000
β	33
γ	2

Table 5.3: Evaluation Parameters – Varying Number of Indices

Figure 5.6 shows the performance of the adaptive OID system during all 3 simulations. Generally, the larger the set of indices, the better the performance of the system after an adaptation, because with increasing number of indices, more queries are able to find an optimal index for resolution. Since more queries are optimized, the overall system performance also improves slightly with increasing number of indices, e.g., the system with 4 indices has 2.3% less messages compared to the system with 3 indices. Similarly, the system with 5 indices has 1% less messages compared to the system with 4 indices.

5.4.3 Varying Number of Data Objects

In this section, we present the performance evaluation of the adaptive OID system by showing the impact of varying number of data objects on the system. We perform 6 different simulations using the same workload as in the first simulation discussed in Section 5.4.1. The parameters used for performing these simulations are shown in Table 5.4. Note that the total number of data objects in the system λ is doubled across the simulations. For each simulation we plot the defined as: average number of simulation time units needed for an adaptation to happen in the system.

Parameter	Value
N	1000
n	1600
o	3
λ	5k, 10k, 20k, 40k, ..., 160k
β	33
γ	2

Table 5.4: Evaluation Parameters – Varying Number of Data Objects

Figure 5.7 shows the performance of the adaptive OID system with respect to the average adaptation window size. The larger the number of data objects in the system, the longer it takes for an adaptation to be triggered. This happens because with increasing number of data objects in the system, the index installation cost $cost_{in}$ also

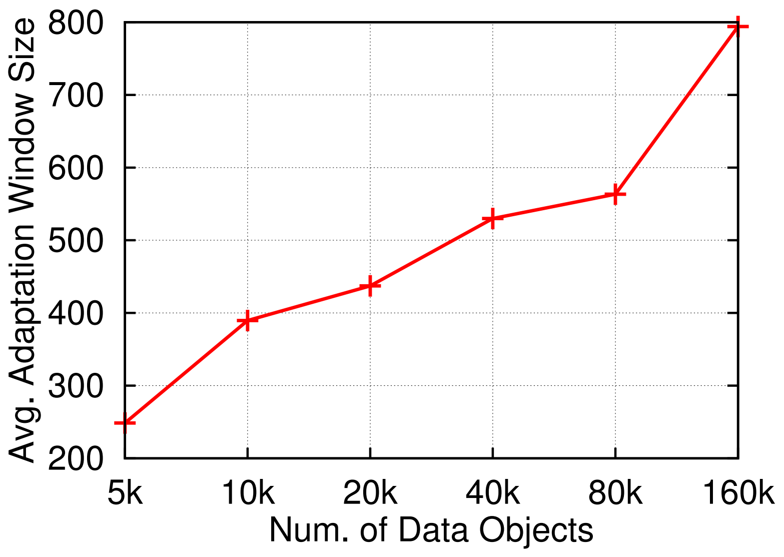


Figure 5.7: Varying Number of Data Objects

increases. Therefore, a larger and more diverse workload of queries is needed for the installation of a new set of indices to be beneficial.

5.4.4 Efficiency of Distributed Workload Collection

In this section, we discuss the results from the performance evaluation of the distributed workload collection (see Section 5.3.1) phase of the index adaptation process. We perform 16 different simulations using the same workload as in the first simulation discussed in Section 5.4.1. For a fixed DHT network size, we vary the values of β and γ across 4 simulations, such that the total number of peers sampled in the network vary between 6% and 12% (in steps of 2%) of the total network size. This simulation scenario is repeated for varying DHT network sizes of $N = (10^2, 10^3, 10^4, 10^5)$. Other simulation are shown in Table 5.5.

Parameter	Value
n	1600
o	3
λ	5000

Table 5.5: Evaluation Parameters – Efficiency of Distributed Workload Collection

During each simulation, after a distributed workload collection phase ends, we measure the cost deviation metric defined as:

$$\left(\frac{|cost(W, I_r^{SW}) - cost(W, I_r^{W})|}{cost(W, I_r^{W})} \right) * 100$$

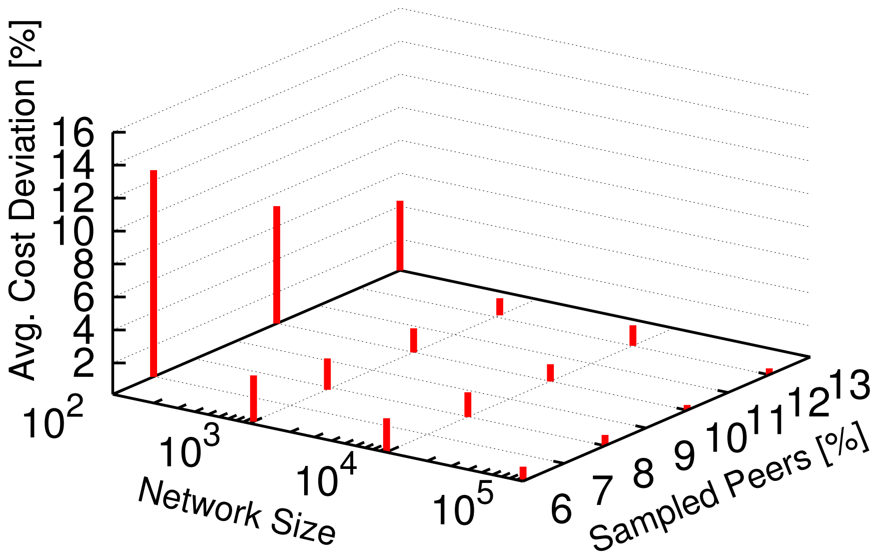


Figure 5.8: Efficiency of Distributed Workload Collection

where W is the complete set of multi-attribute range queries from all peers in the system, I_r^{SW} is the recommended set of indices obtained using the sampled workload SW, and I_r^W is the recommended set of indices obtained using the complete set of queries W.

The cost deviation metric indicates how good the recommended set of indices is (in percentage) if it is obtained using the sampled workload, compared to the recommended set of indices obtained using the global workload. The lower the value of cost deviation, the better the performance of the system because the indices are more optimized for future queries in the system.

For each simulation, the average cost deviation is plotted in Figure 5.8. For the network size of 10^2, the calculation for the number of peers to sample using β and γ, was rounded-off to the same value (7% of the network size) in case of 6% and 8% sampled peers.

Figure 5.8 shows that for a fixed network size, the larger the number of sampled peers, the smaller is the cost deviation. This happens because with increasing number of sampled peers, a better approximation of the complete set of queries is acquired. Hence, the recommended set of indices obtained using the sampled workload is more similar to the recommended set of indices obtained using the complete set of queries. Figure 5.8 also portrays that with increasing network size, sampling a smaller percentage of peers in the network is sufficient for having a low cost deviation.

5.5 Related Work

A number of adaptive P2P information discovery systems have been proposed in the past. In this section, we discuss some of them in relation to our system.

5.5.1 Unstructured P2P Information Discovery Systems

Several unstructured P2P information discovery systems have been suggested that improve the efficiency of future queries based on the past query workload in the system [AC08, KGZY02, KPPT05, NYF08]. The major difference between these systems and the structured P2P systems such as ours is that, each peer in these systems tries to optimize the performance of queries individually by modifying local data index. This does not necessarily lead to the optimization of overall system performance. Moreover, given a query, these systems perform only a best-effort search in the network, i.e., not all matching data objects are always retrieved.

Acosta et al. [AC08] create a synopsis of neighboring peer content at each peer according to the popular terms in queries as well as the content. They show that the search efficiency of such synopsis is better than the synopsis created only using the popular terms in content. Kalogeraki et al. [KGZY02] introduce an intelligent search mechanism where each peer maintains a profile of all its neighbors based on previously answered queries. Later, these profiles are used to determine the peer that is most likely to answer a query. Koloniari et al. [KPPT05] cluster peers in an overlay network based on the type of query workload that the peers are able to process efficiently. They show that given a query, such clustering of peers is able to retrieve more results than simple content-based clustering. Nguyen et al. [NYF08] improve the keyword-based grouping of documents at a peer using popular keywords in queries. They show that using their grouping technique, queries are able to retrieve more documents than simple content-based grouping.

5.5.2 Structured P2P Information Discovery Systems

In order to improve the search efficiency of queries in structured P2P information discovery systems, several DHT extensions have been proposed [DFZ$^+$09, DTK08, SA06].

Deng et al. [DFZ$^+$09] introduce learning-aware blind search for range queries in DHTs. Each peer in their system stores information about previously retrieved results from each link of the DHT using a local index structure. Queries are forward to regions of the DHT that had previously returned the highest number of results. Unlike our system, their system performs only best-effort search since each peer tries to optimize the query performance individually.

Skobeltsyn et al. [SA06] present a system that stores the results of frequently issued queries at certain peers in the DHT. The choice of queries whose results are cached is based on the dynamic workload of queries in the system. A query is resolved first by looking up the results in the local cache. If no results are found, the peer tries to find a neighboring cache with results. If still no results are found, the query is sent to all peer using broadcast. In our system, queries are never resolved using broadcast since it is highly unscalable to resolve queries in such manner. Instead, we optimize indices for efficient query processing.

The HiPPIS system [DTK08] indexes the data in a DHT using hierarchical indices. Each peer in the system logs each query that it issues. If the granularity of a queried attribute changes locally at a peer, e.g., more queries contain "`city`" attribute instead of the "`state`" attribute, the peer checks if the index has to be adapted accordingly. The peer performs the adaptation check by asking every peer in the system for the query statistics on the attribute using flooding. If adaptation is needed, the peer locks all the peers in the system by flooding a lock message. During this period, queries are answered also using flooding. Finally, the adaptation message is sent to all peers in the system using flooding as well. Unlike the HiPPIS system, our system has a flooding-free scalable index adaptation process.

5.6 Conclusion

In this chapter, we presented the design and evaluation of the adaptive OID system. The adaptive OID system optimizes the overall system performance for multi-attribute range queries by dynamically adapting the set of indices in the system. The set of indices is adapted using a four-phase index adaptation process. During the first phase, a workload of multi-attribute range queries is collected from the DHT network using uniform random sampling of peers. This workload is then used in the second phase for obtaining a new set of indices using an index recommendation algorithm (see Chapter 4). During the third phase the cost and the benefit of installing a new set of indices is estimated. If it is beneficial to install the new set of indices, the installation is carried out during the fourth phase of index adaptation process.

Our evaluations show that the adaptive OID system continuously adapts the set of indices in the system according to the dynamic workload of multi-attribute range queries. The adaptations are most useful when there is a variety of different queries in the system. Nonetheless, the adaptive OID system shows several orders of magnitude improved performance compared to a non-adaptive system.

Chapter 6

Application: Spatial Information Discovery

6.1 Introduction

Peer-to-peer (P2P)-based spatial information discovery systems require support for location-based range queries. The support for these queries can be provided either by constructing a specialized P2P overlay network or by utilizing a Distributed Hash Table (DHT)-based P2P overlay network.

P2P spatial information discovery systems, such as Globase.KOM [KLS07], GeoPeer [AR04], and [KLL04] construct a specialized network topology by organizing peers in a tree or a lattice structure to support location-based range queries. Due to complex structural requirements, the network construction and maintenance overhead for these systems is in general higher than that of a DHT with a simple basic structure such as Chord or CAN (see Section 2.2.2).

Conventional DHT-based P2P overlay networks, such as Chord (see Section 2.2.2.1), CAN (see Section 2.2.2.2), and Pastry (see Section 2.2.2.5) used to support only exact-match queries. However, recently they have been extended to support multi-attribute and range queries [MTD+08, AX02, CFCS03, GYGM04, SP03, SOTZ05]. P2P spatial information discovery systems, such as [CRR+05, HT03, KW06, THS+04, WZK05] leverage the DHT-based systems by implementing a layer on top of them to support location-based range queries. This layer consists of either a tree-based data index or a locality-preserving mapping of the data to the peers. Such a layered architecture is attractive because it allows any 3^{rd} party DHT implementation to be used without modifications. However, these systems have some shortcomings that are discussed below.

One of the most desirable property of the P2P spatial information discovery systems is the locality preservation of the data, i.e., data objects that lie close on the surface of the earth, should be placed on the same peer or a group of neighbouring peers in the network. This allows for the efficient retrieval of the data objects, without flooding the whole network with a query. P2P spatial information discovery systems that use only a tree-based data index on top a DHT, do not preserve the locality of the data well. Moreover, P2P spatial information discovery systems that utilize a locality-preserving mapping, do not scale because of the large number of messages generated by location-based range queries.

In this chapter, we present a P2P spatial information discovery system based on the architecture of the OID system (see Chapter 3). Our spatial information discovery system performs a scalable resolution of location-based range queries in spite of using a locality-preserving mapping of the data to the peers. Our system has a layered architecture to allow any DHT to be used as a network structure. The only requirement is that the DHT provides the standard *lookup(key)* and *notify()* methods for communication. The key innovations of our system include: (1) use of a less-distorting octahedral map projection to represent the spatial data, (2) utilization of the Sierṕinski Space-Filling Curve (SFC) for locality-preserving mapping of the data to the peers, and (3) a data placement strategy that reduces the probability of the data hot-spots in the network.

The rest of the chapter is organized as follows: in Section 6.2, we discuss the architecture of our system along with its components. Spatial data indexing is described in Section 6.3. In Section 6.4, the placement of the spatial data in a DHT network is discussed. Query resolution and query optimization are described in Section 6.5. Results from the system evaluations are presented in Section 6.6. In Section 6.7, we discuss the related spatial information discovery systems. Finally, we wind up the chapter with a conclusion Section 6.8.

6.2 System Architecture

Since the spatial information discovery system presented in this chapter is based upon the basic OID system, the architecture of our system is the same layered architecture as the one shown in Figure 3.1.

The top layer of the architecture is the application layer that consists of large-scale distributed applications that rely on spatial range queries, e.g., location-based services or other context-aware applications.

The middle layer is the framework layer. The framework layer consists of four components: data index space, data placement controller, query engine, and query optimizer. The data index space consists of several SFC-based indices used for generating identifiers for the spatial data objects. These data objects are then placed in the network by the data placement controller. The query engine of the framework layer is responsible for the query resolution. In order to achieve scalable resolution of spatial range queries, the query optimizer performs the routing optimization.

The bottom layer is the DHT layer. The DHT layer provides the methods *lookup(key)* and *notify()*. Given a key, the *lookup(key)* method returns the IP address of the node responsible for it, while *notify()* is a call-back method that indicates a change in the key set of a node.

6.2.1 Data and Query Model

The spatial data objects in our system are defined as points on the surface of the earth. Each data object consists of two primary attributes, namely `Latitude` and `Longitude`. Apart from these primary attributes, a data object can accommodate several other attributes that describe its properties. For example, a data object can be defined as (`Latitude` = 48.745°, `Longitude` = 9.106°, `Building` = $University$, `Department` = CS, ...). However, it would be indexed only using the primary attributes `Latitude` and `Longitude`.

A spatial range query in our system is defined as a polygon on the surface of the earth. The query polygon can be described as a conjunction of tuples of the form $(\phi = value, \lambda = value)$ where ϕ and λ represent `Latitude` and `Longitude`, respectively. For example, a spatial range query could be defined as $\{(\phi = 48.747°,\ \lambda = 9.106°) \wedge (\phi = 48.743°,\ \lambda = 9.101°) \wedge (\phi = 48.744°,\ \lambda = 9.111°\ )\}$. All the spatial data objects held by the peers that are responsible for the areas corresponding to the query polygon, are returned. A spatial range query can also include other attributes to allow local filtering of data on the queried peers.

6.3 Data Indexing

As discussed earlier, the locality preservation of the spatial data is the most desirable property of spatial information discovery systems. Locality preservation could be achieved by utilizing SFC-based data indexing. Since SFCs work with Cartesian coordinate system, the surface of the earth has to be projected from geographical coordinate system to the Cartesian coordinate system. We utilize the octahedral map projection [Fur97], in contrast to the quadrilateral projections, for this purpose. Figure 6.1 shows the octahedral map projection of the surface of the earth. The Octahedral map projection achieves good polar symmetry and the important global features (poles, equator and meridians) are mapped to triangular faces and vertices of the projection [LT92]. To the best of our knowledge, such a map projection has not been used by any P2P spatial information discovery system.

Although quadrilateral maps are easier to work with, they tend to have greater distortion as we start to move away from the equator towards the poles of the earth [IG01]. Octahedral map projection, however, is less distorting. This means that if the surface of an octahedral globe is sub-divided into smaller regions, each region would be of

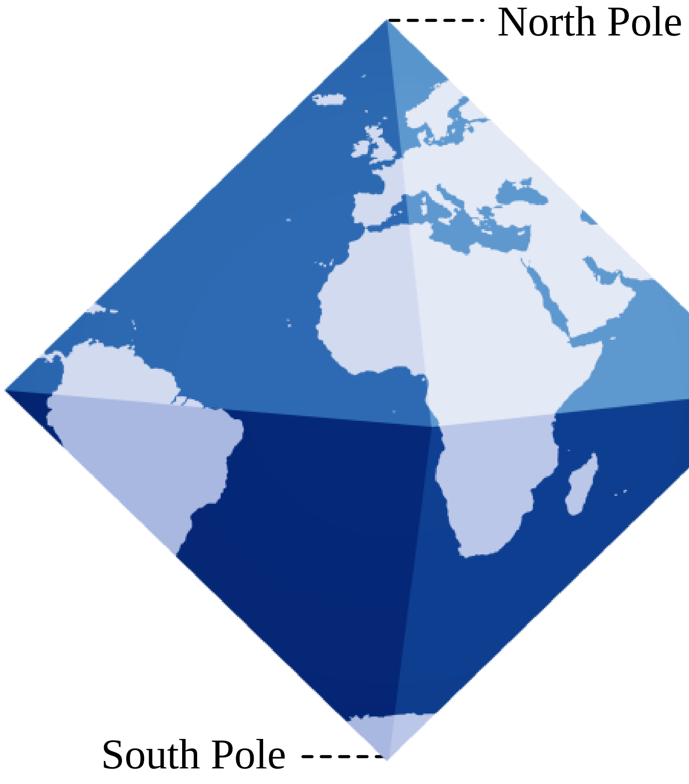


Figure 6.1: Octahedral Map Projection

approximately equal areal shape and size. If these regions are uniformly distributed over a network of peers, each peer would hold information about almost an equal area of the earth, which would not be the case if a quadrilateral map projection is used.

We utilize Sierṕinski SFC [IG01] to map the spatial data objects on the surface of the octahedral map to the DHT while preserving data locality. Sierṕinski SFC is chosen over Hilbert [Hil91], Peano [Pea90], Z [OM84], and other SFCs because it achieves regular sub-divisions of the triangular faces of the octahedral map.

A Space-Filling curve (SFC) is essentially a mapping from a multi-dimensional space to a 1-dimensional line (see Section 2.3). We create eight SFC-based data indices using Sierṕinski SFCs, one for each face of the octahedral map.

Each face of the octahedral map is a triangle. The construction of the Sierṕinski SFC on a triangular face of the octahedral map is a two step process. The first step divides the triangle into smaller triangles (which we call zones). This division step can be viewed as a recursive process, where each run of the process divides each triangle into two equal triangles. The process continues k times yielding 2^k zones. In the next step, a line is drawn that enters and leaves the triangular face of the map, passing through each of the 2^k zones once. The line imposes an order among the zones as it walks through them. As a result, a k^{th} order SFC is defined for the triangular face of the earth. Figure 6.2

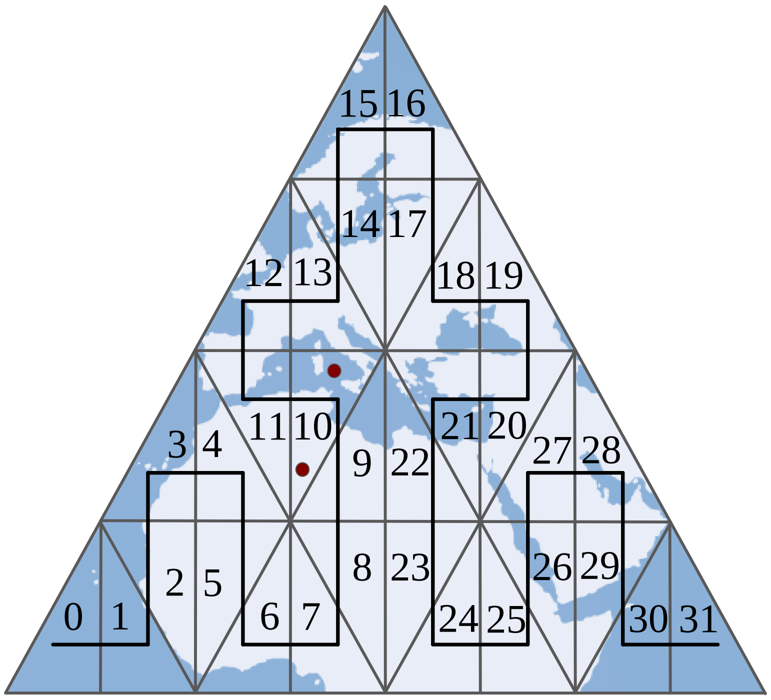


Figure 6.2: Sierṕinski Space-Filling Curve

shows a 5th order Sierṕinski SFC on a triangular face of the octahedral map that covers parts of Asia, Africa, and Europe.

The part of the SFC line that passes through a zone is an approximation of all the points in that zone. Consider the example of the spatial data objects shown as dots in Figure 6.2. Although the data objects belong to separate locations on the map, both are indexed by the 10th zone on the Sierṕinski SFC, i.e., both the data objects have an identifier of 10.

Even though a single continuous Sierṕinski SFC could be constructed on the octahedral map that indexes all the spatial objects, we utilize a separate SFC-based data index for each face of the map. This is done to achieve better distribution of data over the DHT, which we discuss in the next section.

6.4 Data Placement

The data placement controller in the framework layer of our system is responsible for placing the spatial data objects over a DHT. Without loss of generality, we utilize the Chord DHT (see Section 2.2.2.1) to place spatial data objects in the network.

When the OID framework layer of a peer receives a new data object from an application, the data placement controller generates an identifier for it using the Sierṕinski SFC (see Section 6.3). Once the identifier for the data object is known, the data placement controller performs a DHT lookup using this identifier in order to determine the peer that is responsible for the data object. Finally, the spatial data object is directly transferred to the peer responsible for hosting it.

The DHT layer of a peer reports any changes in the data set of the peer to the data placement controller using the *notify()* method call. These changes occur when the DHT performs a repair operation after a peer joins or leaves the network. After reception of a notification, the data placement controller redistributes the data among the neighbours accordingly.

If a single Sierṕinski SFC is used to index all the spatial data objects on the surface of the octahedral map, the result could be a non-uniform distribution of the data over the Chord ring. For example, if a system indexes all the restaurants in the world, then the peers responsible for holding information about the oceans would contain no data. Since oceans are the largest parts of our planet, the majority of the peers would not be utilized for storage. To reduce the degree of non-uniform distribution of data, we use a separate SFC for each face of the octahedral map. This way, a peer is responsible for eight different regions on the map and the probability that each of these region contains no data, is reduced.

Using a separate SFC for each face of the octahedral map comes with a trade-off, i.e., the locality of the data along the edges where two faces meet, is disturbed. For example, if a single SFC is used, two spatial data objects that lie along the edge where two faces meet, would receive data identifiers that are numerically close to each other. On the contrary, the same data objects would receive very different identifiers if two separate SFCs are used to index the two faces. Therefore, it is not straight-forward to extend this approach for map projections with larger number of faces, e.g., dodecahedral and icosahedral map projections.

Typically, the identifier space of the Sierṕinski SFC, $[0, 2^k)$, is smaller than the identifier space of the Chord ring, $[0, 2^m)$. Therefore, if the spatial data objects from a face of the octahedral map are directly placed on the Chord ring, the peers with identifier values greater than 2^k, would contain no data. To avoid this, we scale up the identifier space of the SFC by a factor of 2^{m-k}. Details and an example of up-scaling could be found in Section 3.5.

6.5 Query Resolution

Once the spatial data objects are placed in the network, they could be retrieved by any peer using a location-based range query. As described earlier, a location-based range query is defined as a polygon on the surface of the earth and all the spatial data objects covered by the area of this polygon are retrieved. The query engine that is a part of the framework layer is responsible for performing the query resolution.

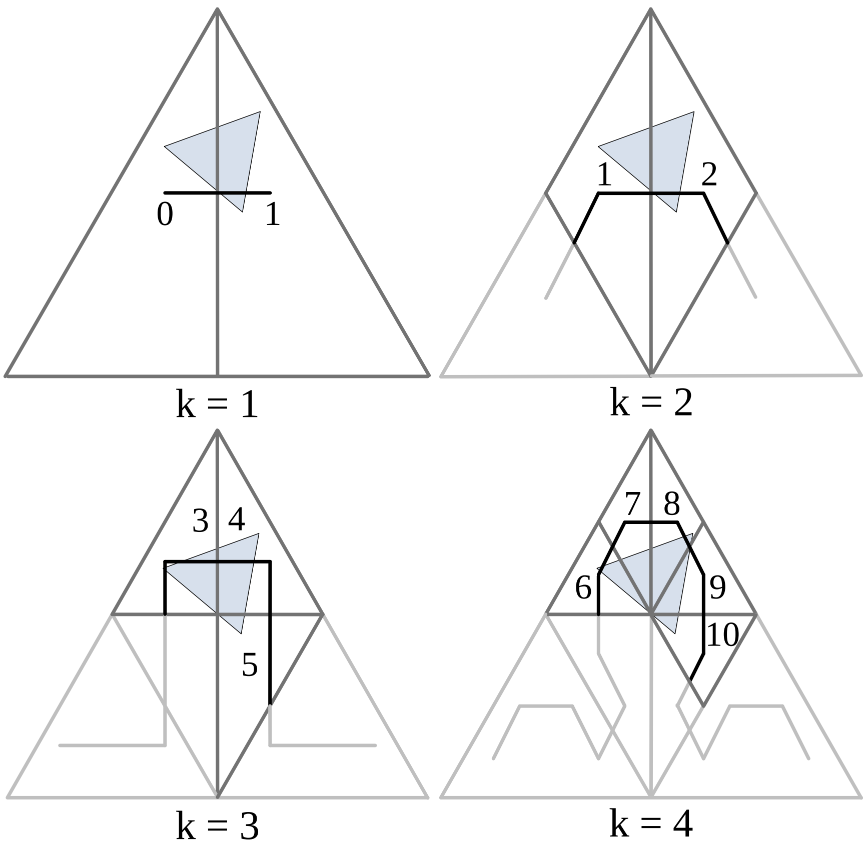


Figure 6.3: Successive Query Refinement

After a location-based range query is received by the query engine from an application, the following sequence of operations are performed by the query engine. First, the queried area is mapped onto the octahedral map in order to determine the triangular faces of the map that are covered by the query. These triangular faces are then further refined using the Sierṕinski SFC up till the highest approximation level k. During the refinement process, the SFC zones that do not match the query polygon are pruned away, resulting in a set of matching zone identifiers. The matching zone identifiers, form clusters of zones where a single cluster contains a sequence of continuous zone identifiers. Once the identifiers of the zones matching the query are determined, the peer can perform either the basic query resolution discussed in Section 3.6 or it can use the query optimization discussed in Section 3.7.

Figure 6.3 shows an example of successive refinement of a query by the Sierṕinski SFC with the maximum approximation level of 4. The query is shown as a shaded triangle (3-sided polygon) and the grayed out areas represent the zones that are pruned away by the refinement process. For the 1^{st} approximation level, the query matches the zones 0 and 1. Therefore, both of these zones are further refined to get four zones in the next approximation level. Note that for $k = 2$, the identifiers for the non-matching zones are

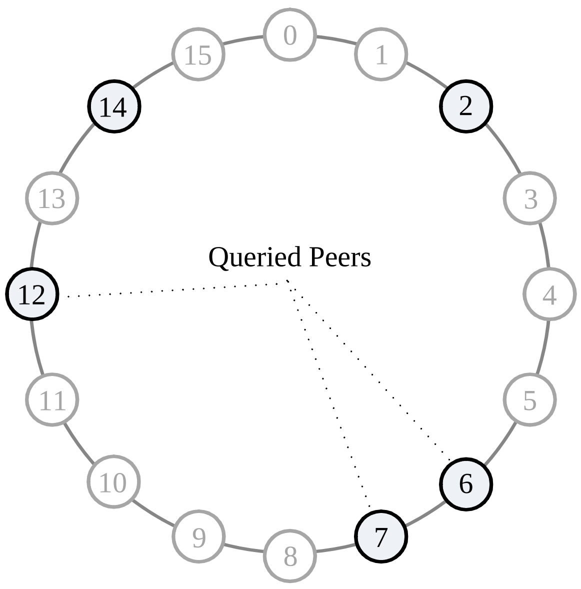


Figure 6.4: Chord Ring with $m = 4$ and $N = 5$

not generated by the refinement process, which indicates that these zones are pruned away. For $k = 3$, only zones 1 and 2 from the previous step are further refined, since only these zones match the query. Finally for the last approximation level, zones 3 to 5 from the previous step are refined further to get zones 6 to 10 as a single cluster.

Now consider a Chord identifier ring shown in Figure 6.4 (long distance links have been removed for clarity). Physical peers are shown as circles with darker borders. The query in the example above would be resolved by the peers 6, 7 and 12.

6.6 System Evaluation

In this section, we present evaluations of our system. We simulated the use of Sierṕinski SFC over the Chord DHT. The simulations were performed over a cluster of 32 nodes with each node having a 2.7 GHz dual-core AMD Opteron processor and 8 GB of RAM.

The data set for our simulations is generated using the NASA Visible Earth [Ear08] image of the city light distribution over the globe (see Figure 6.5), where the probability of bright pixels is higher than the probability of the dark pixels. This image, for example, represents the distribution of developed areas around the world.

For performance comparison, we simulated the following two configurations of our system:

- **Configuration A (Multiple SFCs):** A separate Sierṕinski SFC is used for each face of the octahedral map to index the spatial data.

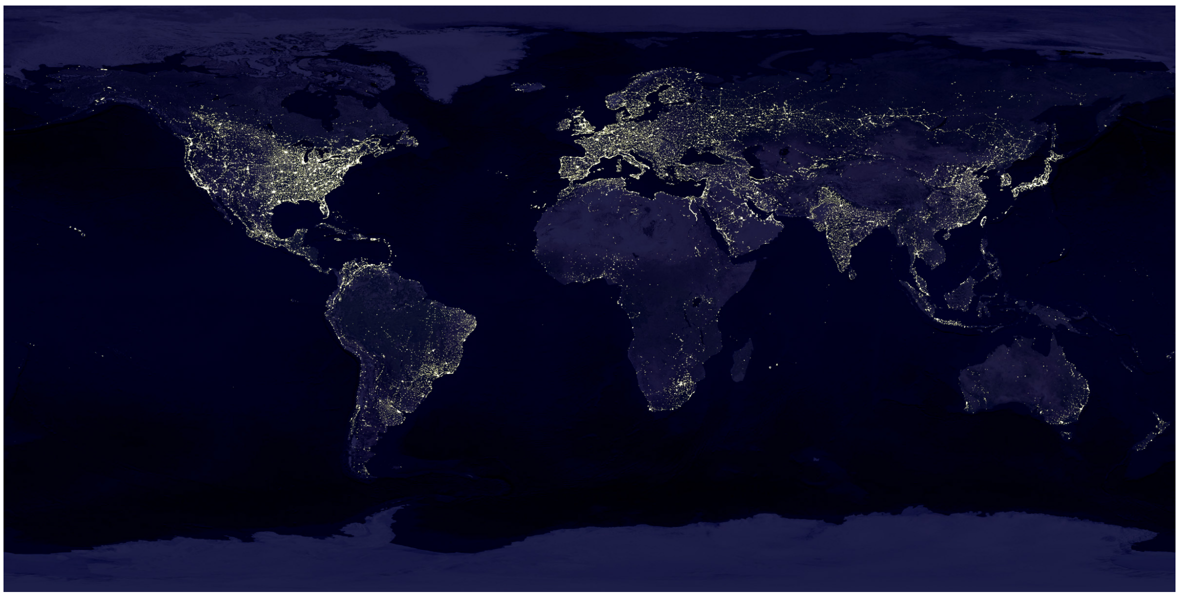


Figure 6.5: NASA Visible Earth Light Distribution

- **Configuration B (Single SFC):** A single continuous SFC is used to index the spatial data on the octahedral map.

Param	**Value(s)**	**Description**
N	$10^2, 10^{2.5}, 10^3 \ldots 10^5$	# of peers
m	128	# of Chord identifier bits
O	$10 * N$	# of spatial data objects
k	32	SFC approximation level
f	10	fanout
l	$\lfloor log(N) \rfloor - 1$	depth-level

Table 6.1: Evaluation Parameters – Performance Evaluation

6.6.1 Performance Evaluation

For evaluating the performance of our spatial information discovery system, we perform a location-based range query over Germany with the parameters shown in Table 6.1. The peer identifiers are uniformly distributed over the identifier space of $(0, 2^m)$. Like a typical P2P system, the total number of data objects is increased with the network size. The approximation level k is chosen such that a zone of the SFC represents an area that is approximately equal to the size of a FIFA football field (approx. 8000 m^2). The values for the query parameters f and l are chosen such that the number of parallel messages is restricted to 10% of the network size in the worst case. The

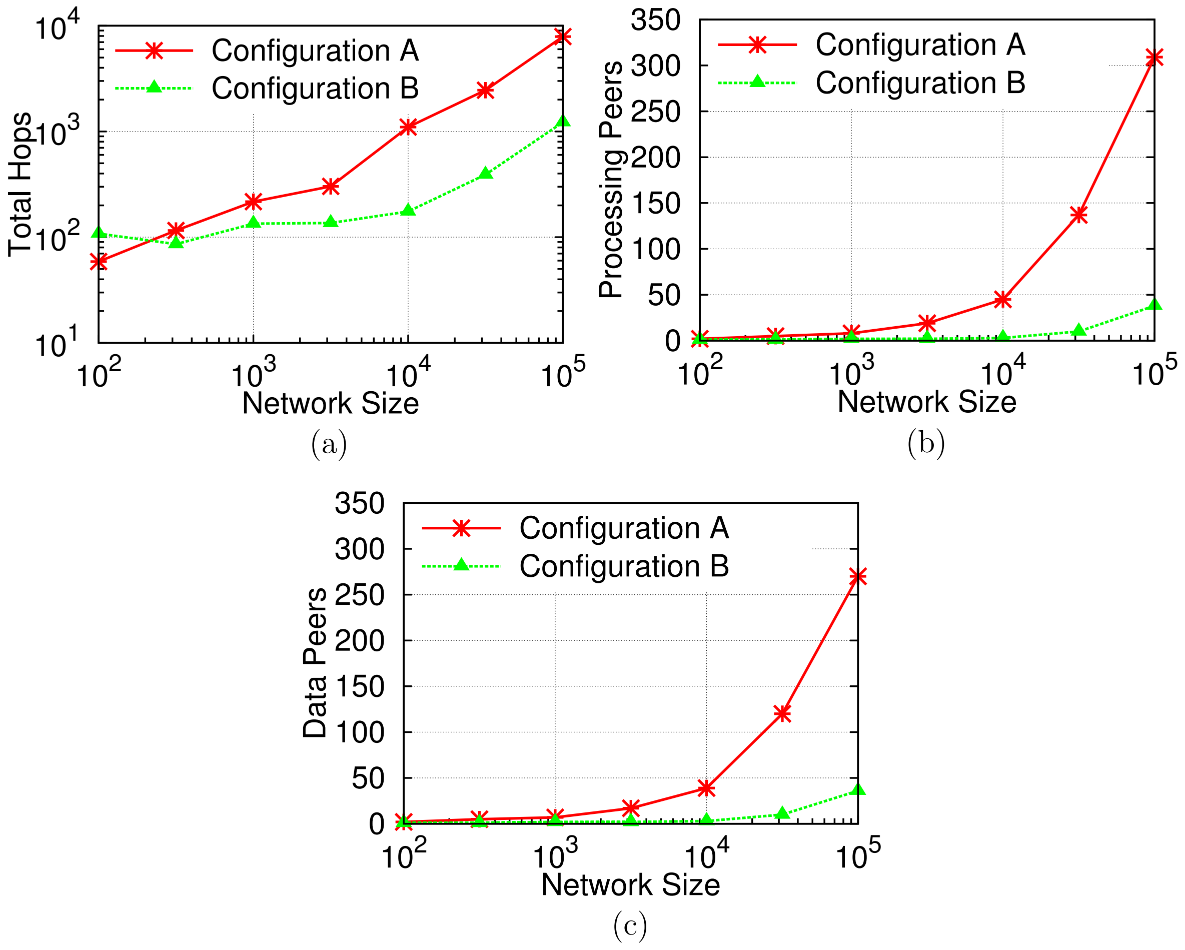
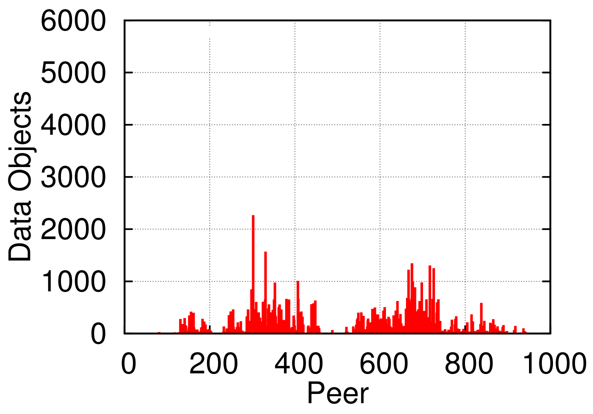


Figure 6.6: Performance of a Spatial Range Query

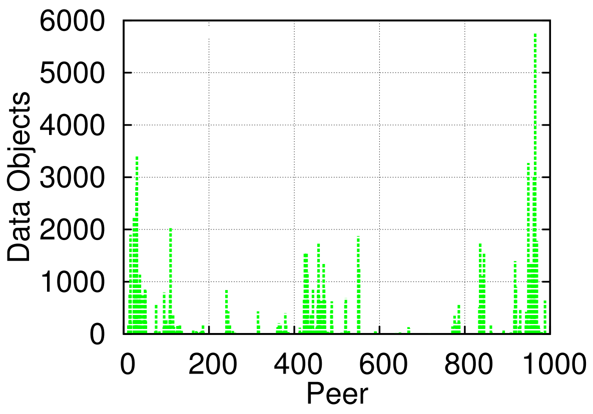
following performance metrics are measured for the two types of system configurations described above:

- **Total Hops** - Total number of hops of all messages needed to resolve a query. This includes the hops of the DHT lookup messages as well.
- **Processing Peers** - Number of peers that perform a local repository lookup to evaluate the query.
- **Data Peers** - Number of peers containing the data objects that match the query (subset of processing peers).

Figure 6.6 shows the performance of both system configurations with respect to the performance metrics defined above. For each of the network sizes shown in the graphs, the location-based range query is performed 10 times starting from a random peer in the network. The values are then averaged to produce a point on a graph. Note that the x-axis of Figure 6.6(a)–(c), and the y-axis of Figure 6.6(c) are on a log scale.



(a) Configuration A (Multiple SFCs)



(b) Configuration B (Single SFC)

Figure 6.7: Data Distribution

The total number of hops, the number of processing peers, and the number of data peers increase with the network size in the case of both system configurations (Figure 6.6(a)–(c)). With the increasing network size, each peer is responsible for smaller range of identifiers. Hence, in order to resolve a location-based range query, the query has to be evaluated at a larger number of peers.

For each performance metric, system Configuration B (Single SFC) performs better than Configuration A (Multiple SFCs), because the spatial data is better distributed over the network in the case of system Configuration A. Therefore, the query has to be evaluated at a larger number of peers compared to the system Configuration B. However, as discussed earlier (see Section 6.4), there is a trade-off between the performance and the degree of data distribution. The degree of data distribution for both system configurations is analysed in the next section.

6.6.2 Data Distribution

In this section, we analyse the data distribution properties of the two different configurations of our spatial information discovery system defined above. The simulations are performed using the parameters shown in Table 6.2. For each peer in the network, the following metric is measured in the case of both system configurations:

Param	**Value(s)**	**Description**
N	10^3	# of peers
O	10^4	# of spatial data objects
k	32	SFC approximation level

Table 6.2: Evaluation Parameters – Data Distribution

- **Data Objects** - Number of data objects that a peer stores in its local repository.

Figure 6.7 shows the degree of data distributions achieved by the two system configurations. Clearly, there are several data hot-spots in the case of system Configuration B (Single SFC) (Figure 6.7(b)). However, Figure 6.7(a) shows that the spatial data is better distributed in the case of system Configuration A (Multiple SFCs). For example, the number of peers responsible for 1000 or more data objects is 7 for system configuration A, compared to 24 for system configuration B.

The difference in the data distribution of the two system configurations is due to the fact that the peers responsible for holding information about the oceans contain no data in the case of system Configuration B (as there is hardly any light in oceans, see Figure 6.5). However, in the case of system Configuration A, a peer is responsible for holding information about 8 different regions of the world. Therefore, the probability of a peer containing no spatial data is reduced drastically.

6.6.3 Performance vs. Data Distribution

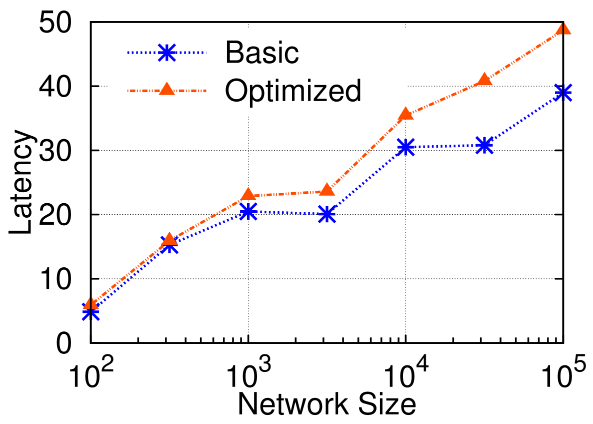
As shown by the simulation results in Section 6.6.1 and Section 6.6.2, there is a trade-off between performance and the degree of the data distribution over the network. If the spatial data is non-uniformly distributed over the ranges of latitude and longitude (like with our data set), the system configuration with a separate SFC for each face of the octahedral map achieves better distribution of the data over the network, compared to the system configuration with a single continuous SFC. However, the performance deteriorates.

The query used for the performance evaluation is over the area of Germany. However, location-based range queries are typically over smaller areas, e.g., a single city. Since the performance of the system configuration with a separate SFC for each face of the octahedral map in not extremely inefficient, it is still feasible to utilize such a system configuration. For example, a query over the city of Stuttgart in Germany is resolved in 230 overlay hops over a network of 10^5 peers by the same system configuration.

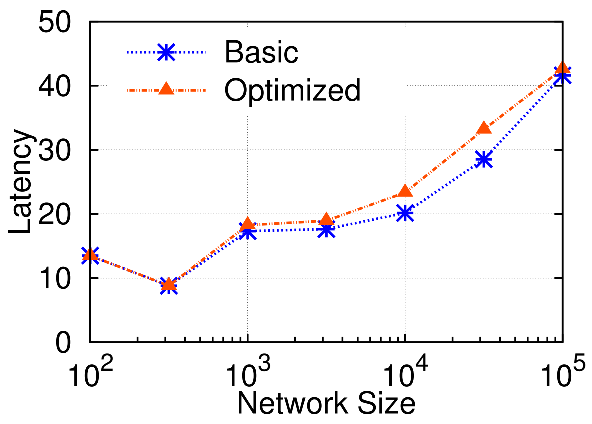
6.6.4 Query Optimization

In this section, we discuss the results of several experiments performed to analyse the performance of the system using the basic and optimization query routing algorithms (see Section 6.5). Each experiment is performed on both types of system configurations defined above. The following performance metrics are monitored during the experiments:

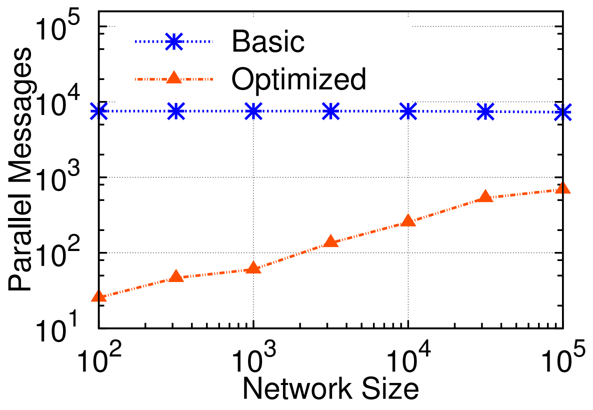
- **Latency** - Number of hops of the longest message path.
- **Average Number of parallel messages** - Total number of forwarded messages divided by the number of levels of the search tree.



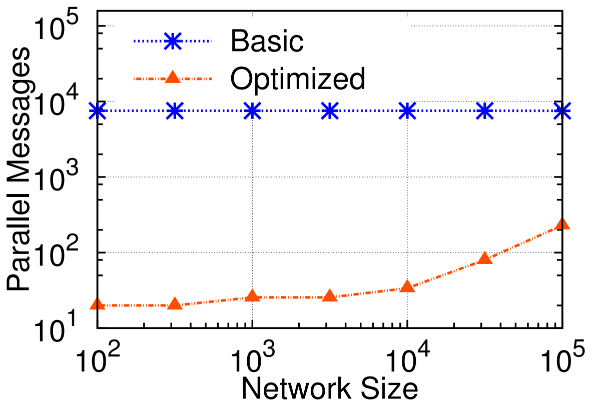
(a) Configuration A (Multiple SFCs)



(b) Configuration B (Single SFC)



(c) Configuration A (Multiple SFCs)



(d) Configuration B (Single SFC)

Figure 6.8: Basic Routing vs. Routing Optimization (Experiment 1)

The first experiment portrays the effects of varying network size on the performance of the two query resolution algorithms with respect to the metrics defines above. The simulation parameters are shown in Table 6.3.

Param	**Value(s)**	**Description**
N	$10^2, 10^{2.5}, 10^3 \ldots 10^5$	# of peers
O	$10 * N$	# of spatial data objects
k	32	SFC approximation level
f	10	fanout
l	3	depth-level

Table 6.3: Evaluation Parameters – Basic Routing vs. Routing Optimization (Experiment 1)

Figure 6.8(a)–(b) show that for each of the system configurations, the latency of the query resolution increases with the network size, in the case of both query resolution algorithms. Since with the increasing number of peers in the network, the range of identifiers that a single peer is responsible for decreases, the same query has to be evaluated at a larger number of peers.

The difference between the latencies of the two query resolution algorithms is very small for both types of system configurations (see Figure 6.8(a)–(b)). However, there is a significant difference between the average number of parallel messages produced by the two query resolution algoritms (see Figure 6.8(c)–(d)). The optimized query resolution algorithm produces fewer parallel number of messages than the basic query resolution algorithm, even with the network size as high as 10^5.

The performance of system Configuration B (Single SFC) is in general better than the performance of system Configuration A (Multiple SFCs) because the spatial data is better distributed over the network in the case of system Configuration A. Therefore, the query has to be evaluated at a larger number of peers compared to the system Configuration B.

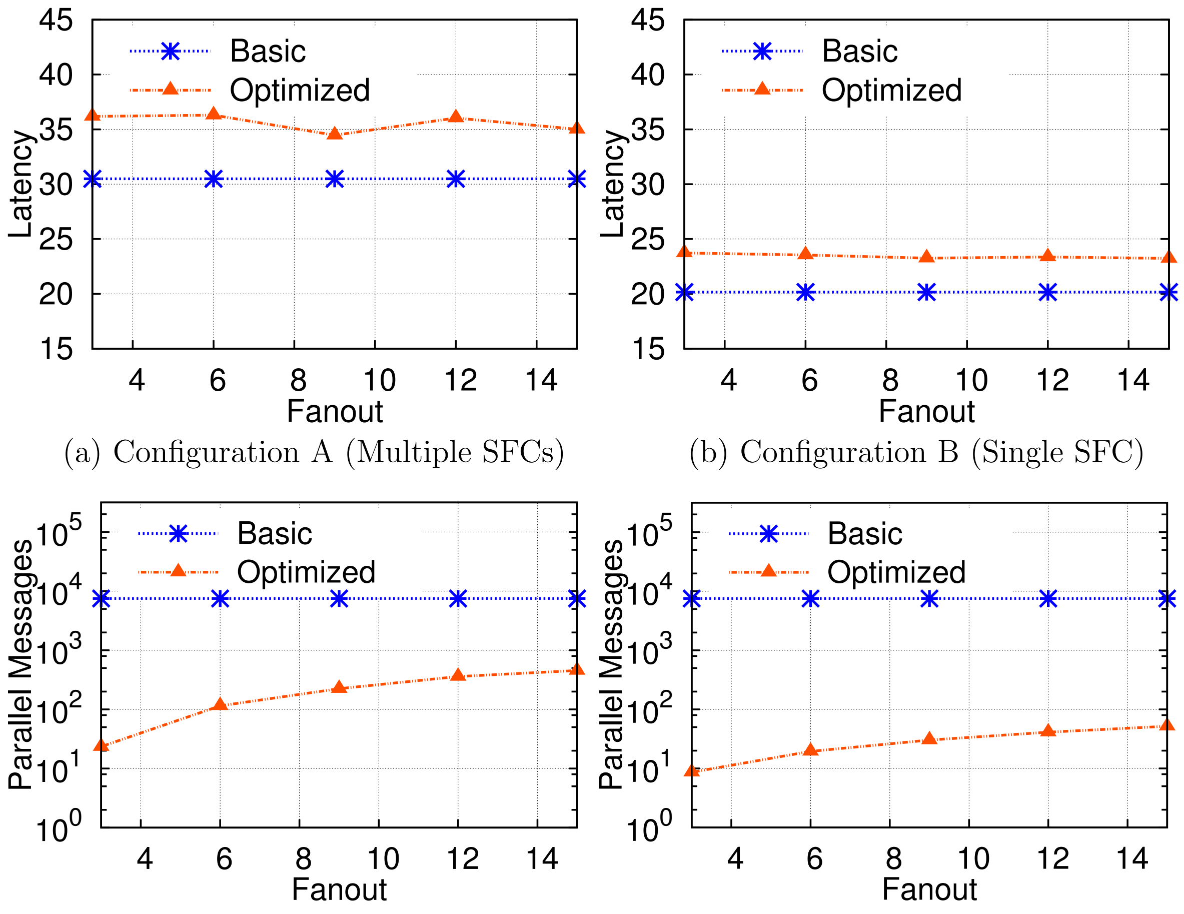
Param	**Value(s)**	**Description**
N	10^4	# of peers
O	10^5	# of spatial data objects
k	32	SFC approximation level
f	3, 6, 9 . . . , 15	fanout
l	3	depth-level

Table 6.4: Evaluation Parameters – Basic Routing vs. Routing Optimization (Experiment 2)

The second experiment shows the effects of varying values of parameter f on the query optimization algorithm, with respect to the performance metrics defined above. For comparison, the performance of the basic query resolution algorithm is also plotted. The second experiment is performed using the simulation parameters show in Table 6.4 for both types of system configurations.

The latency for the query resolution remains constant for the basic query resolution (Figure 6.9(a)–(b)) throughout Experiment 2, because all the query clusters are immediately forwarded by the query initiator to the responsible peers. Due to the same reason, the basic query resolution has slightly lower latency compared to the optimized query resolution.

Although the number of parallel messages for the optimized query resolution algorithm increases with increasing fanout values (Figure 6.9(c)–(d)), it still remains below the number of parallel messages transmitted by the basic query resolution algorithm, even for the fanout value as high as 14. The number of parallel messages increases with the increasing fanout because the query clusters are divided more at each level of the query tree.



(a) Configuration A (Multiple SFCs) (b) Configuration B (Single SFC)

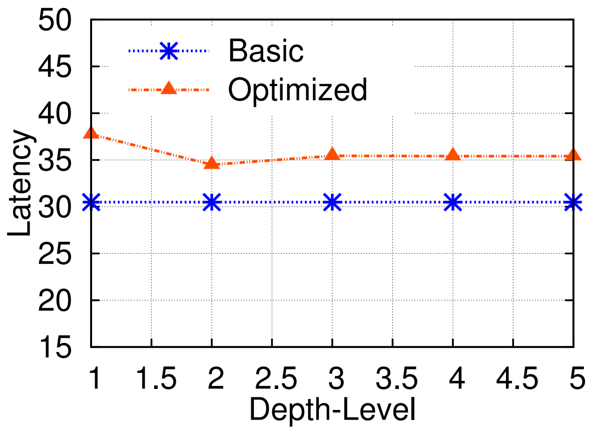
(c) Configuration A (Multiple SFCs) (d) Configuration B (Single SFC)

Figure 6.9: Basic Routing vs. Routing Optimization (Experiment 2)

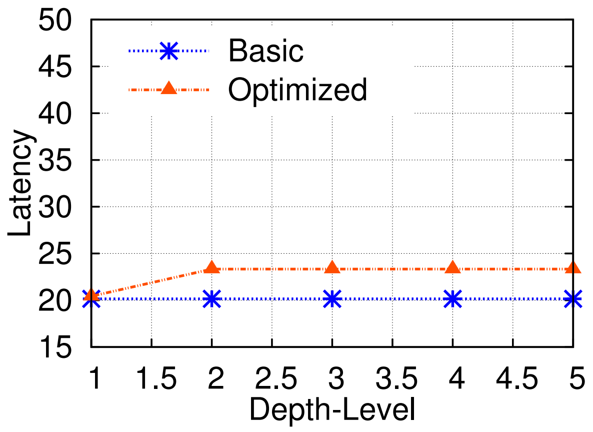
The third and the final experiment shows the effects of varying depth-level parameter l on the query resolution latency and the parallel number of messages in the network. The third experiment is performed using the parameters shown in Table 6.5 for both types of system configurations.

Param	**Value(s)**	**Description**
N	10^4	# of peers
O	10^5	# of spatial data objects
k	32	SFC approximation level
f	10	fanout
l	$1, 2, \ldots, 5$	depth-level

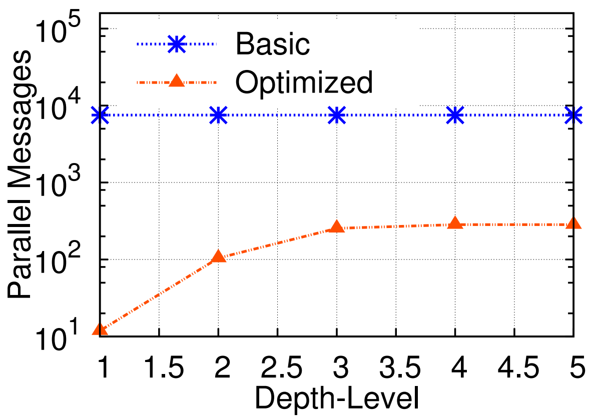
Table 6.5: Evaluation Parameters – Basic Routing vs. Routing Optimization (Experiment 3)



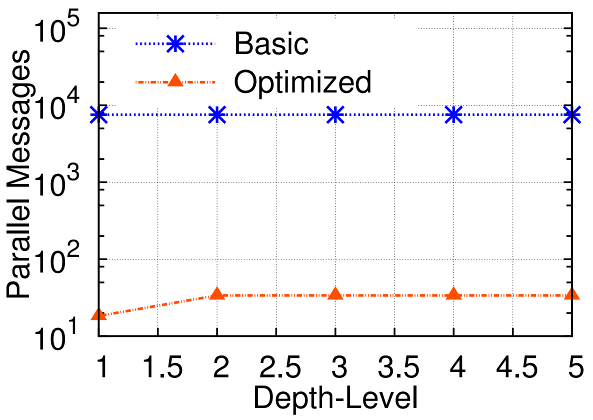
(a) Configuration A (Multiple SFCs)



(b) Configuration B (Single SFC)



(c) Configuration A (Multiple SFCs)



(d) Configuration B (Single SFC)

Figure 6.10: Basic Routing vs. Routing Optimization (Experiment 3)

For the system Configuration A (Multiple SFCs), the query resolution latency first decreases, and then it remains almost constant for the optimized query resolution algorithm (Figure 6.10(a)). In the beginning the latency decreases as more parallel messages are used to resolve the query. Therefore, each message has to travel less through the network. Later, it seems that the whole query is resolved within two levels of the search tree. Therefore, starting from the depth-level 3 onwards, the query resolution latency remains almost constant.

Similarly, for the system Configuration B (Single SFC), the latency remains constant throughout the experiment for the optimized query resolution (Figure 6.10(b)). For $l = 1$, the value for the latency is an outlier because the query initiator was itself responsible for resolution of some query clusters and therefore, did not transmit them through the network.

The latency of the basic resolution algorithm is better than the latency of the optimized algorithm for the same reasons discussed in the second experiment.

The parallel number of messages first increases then remains constant with increasing values of parameter l for the optimized query resolution algorithm (Figure 6.10(c)–(d)). As the value of parameter l increases, more number of messages are generated in order to resolve the query, until a certain point is reached. After this point, more messages cannot be generated because the query gets resolved with fewer messages already.

The parallel number of messages generated by the optimized resolution algorithm is significantly lower than the messages generated by the basic resolution algorithm, because of the same reasons discussed in the first experiment.

6.7 Related Work

Generally, P2P spatial information discovery systems could be divided into two major categories: ones that built a specialized network topology (Non-DHT based Systems) and others that use a DHT-based P2P overlay network to support location-based range queries (DHT based Systems). In this section, we briefly discuss these systems in relation to our spatial information discovery system.

6.7.1 Non-DHT based Systems

The Globase.KOM system [KLS07] enables location-based range queries by organizing peers in a tree-based overlay network. The system utilizes the Plate Carrée map projection to index spatial data on the surface of the earth. Such a map projection introduces more distortion compared to the octahedral map projection utilized by our system. Moreover, the search efficiency of the Globase.KOM system is influenced by the amount of stored cache information which is not the case for our system.

The GeoPeer system [AR04] organizes peers to form a Delaunay triangulation, with each peer being responsible for holding information about a certain area on the globe. Unlike a DHT, the search algorithm opted by the GeoPeer is not as efficient, because each peer maintains a list of only few neighbouring nodes.

The spatial information discovery system introduced by Kang et al. [KLL04] also uses Delaunay triangulation to construct a network of peers. Similar to the GeoPeer system, the network coverage of a spatial query is not limited, i.e., a query could get flooded throughout the network before arriving at the destination peer. In contrast to that, our system uses DHT for routing queries. DHT has a known bound of $O(logN)$ steps to route a message from a source node to a destination node.

6.7.2 DHT-based Systems

DHT-based P2P overlay networks have evolved over a period of time. As a result, scalable implementations of DHTs, such as Open Chord [LK06], are widely available, which raises the desire to utilize them for spatial information discovery.

Harwood et al. [HT03] and Tanin et al. [THS+04] extend the Chord (see Section 2.2.2.1) protocol for spatial information discovery using a quadtree and an octree, respectively. Their systems recursively sub-divide the virtual world into regions and assign these regions to a network of Chord peers. A location-based range query is first translated into a set of identifiers, then a lookup operation is performed for each of these identifiers to resolve the query. The shortcoming of these systems is that the assignment of the regions to the peers is not locality-preserving, leading to longer path lengths in the DHT for the location-based range queries.

P2P spatial information discovery systems proposed by Chawathe et al. [CRR+05], Wang et al. [WZK05] and Knoll et al. [KW06] are the ones that come closest to our system.

Chawathe et al. [CRR+05] use the Z-curve (aka Morton-order or z-order) [OM84] to reduce the multi-dimensional location data to a single dimension. This 1-dimensional data is then distributed over the Chord ring using a Prefix Hash Tree (PHT) [RHRS04]. The use of PHT introduces extra maintenance overhead for the system because the information about the PHT nodes has to be maintained at each peer in the network. In contrast to their system, the only information maintained by our system is the one maintained by the routing tables of the DHT nodes.

Wang et al. [WZK05] sub-divide the space of a CAN (see Section 2.2.2.2) DHT into scatter regions. A scatter region is further partitioned into zones. A spatial data object is then mapped to a single scatter region and hashed to a random zone within the scatter region. The hashing is done to achieve better load balancing, but the locality of the data is lost, particularly if the scatter region is large.

Knoll et al. [KW06] compare the performance of the Peano [Pea90], Z-Curve, and the Hilbert [Hil91] SFC-based indices over a Pastry (see Section 2.2.2.5) network. They analyse the locality preservation properties of each SFC, but do not focus on location-based range queries. Moreover, they utilize a quadrilateral map projection in contrast to the less-distorting octahedral projection used by our system.

Although general DHT-based P2P information discovery systems, such as [GYGM04] and [SP03] could be extended to support location-based range queries, such an extension would require a map projection with only small distortion, like the octahedral projection utilized by our system. Moreover, related systems based on DHTs and SFCs could benefit from the optimized query processing mechanisms discussed in this chapter.

6.8 Conclusion

In this chapter, we presented the system design and evaluation of a DHT-based spatial information discovery system. Our system is the first one to utilize the less-distorting octahedral map projection compared to the quadrilateral projections utilized by the majority of the previously proposed spatial information discovery systems. Our system uses Sierṕinski SFC in order to map the spatial data on the surface of the octahedral map to a network of Chord peers, while preserving the data locality. We proposed and evaluated two different types of system configurations for utilizing the Sierṕinski SFC. The first configuration uses a separate SFC for each face of the octahedral map and therefore achieves a better distribution of the data over the peers. The second configuration uses a single continuous SFC for the complete octahedral map and therefore achieves better query resolution performance.

We increased the scalability of our system by utilizing an optimized query resolution algorithm. According to our simulations, the optimized query resolution algorithm achieves up to 96% reduction in the average number of parallel messages used to resolve a query, equal to the area of Germany, over a network of $100,000$ peers.

Chapter 7

Summary and Future Work

In this chapter, we present the summary of the contributions made during our research. Moreover, a discussion on potential future work is also a part of this chapter.

7.1 Summary

In this book, we introduced the optimized information discovery (OID) system that significantly optimizes the performance of multi-attribute range queries in distributed hash table (DHT)-based peer-to-peer (P2P) overlay networks. Following are the main contributions and results of our research:

- We introduce a novel data indexing approach that uses multiple space-filling curve (SFC)-based indices to index each data object in the system (see Chapter 3). Using this approach, the OID system resolves a multi-attribute range query by selecting the index that yields the best performance for the given query. The best performing index is chosen by estimating the number of peers the query would be evaluated at, using each index. We also introduced two types of query optimizations that improve the scalability of the OID system. The routing optimization limits the message forwarding load of a peer and the number of parallel messages at a time in the system. The computation load distribution algorithm distributes the computation load for the query resolution over several peers in order to avoid bottleneck at particular peers.

  The evaluations of the OID system show that the best performance for multi-attribute range queries is achieved over corresponding completely matching SFC-based indices. Moreover, we show that the routing optimization algorithm reduces the total number of parallel messages in the network with a cost of little increased latency. It is also shown that the computation load distribution algorithm reduces the amount of computation performed by a single peer up to 99%.

- We present several index recommendation algorithms that assist the designer of a distributed application in order to define useful indices for efficient query resolution (see Chapter 4). Given a limit for the maximum number of indices and a workload of queries, each algorithm recommends a set of indices that produces close-to-optimal performance for the workload queries within the given limit. The set of index recommendation algorithms includes three scalable index recommendation algorithms: cost-based merge, similarity-based merge and selection algorithm.

  Our evaluations show that there is a trade-off between the performance and the execution time of the scalable index recommendation algorithms. With respect to the performance, the cost-based merge algorithm is the best (only 1.5% worse than the optimal naïve algorithm), generally followed by the similarity-based merge and the selection algorithms. With respect to the execution time of the algorithms, the order is reversed.

- We introduce an approach for continuously optimizing the overall system performance for multi-attribute range queries by dynamically adapting the set of indices in the system (see Chapter 5). The set of indices is adapted using a four-phase index adaptation process. During the first phase, a workload of multi-attribute range queries is collected from the DHT network using uniform random sampling of peers. This workload is then used in the second phase for obtaining a new set of indices using an index recommendation algorithm. During the third phase the cost and the benefit of installing a new set of indices is estimated. If it is beneficial to install the new set of indices, the installation is carried out during the fourth phase of index adaptation process.

  Our evaluations show that the adaptive OID system continuously adapts the set of indices in the system according to the dynamic workload of multi-attribute range queries. The adaptations are most useful when there is a variety of different queries in the system. Nonetheless, the adaptive OID system shows several orders of magnitude improved performance compared to a non-adaptive system.

- Finally, we show the usability of the OID system by designing and evaluating a spatial information discovery system based on the OID architecture (see Chapter 6). Our spatial information discovery system is the first one to utilize the less-distorting octahedral map projection compared to the quadrilateral projections utilized by the majority of the previously proposed spatial information discovery systems. Our system uses Sierṕinski SFC in order to map the spatial data on the surface of the octahedral map to a network of Chord peers, while preserving the data locality. We proposed and evaluated two different types of system configurations for utilizing the Sierṕinski SFC. The first configuration uses a separate SFC for each face of the octahedral map and therefore achieves a better distribution of the data over the peers. The second configuration uses a single continuous SFC for the complete octahedral map and therefore achieves better query resolution performance.

We increased the scalability of our system by utilizing an optimized query resolution algorithm. According to our simulations, the optimized query resolution algorithm achieves up to 96% reduction in the average number of parallel messages used to resolve a query that covers the area of Germany, in a network of $100,000$ peers.

7.2 Future Work

Currently, for the adaptive OID system, the complete log of multi-attribute range queries is retrieved from a peer during distributed workload collection phase (see Section 5.3.1). In future, this phase could be changed so that it is possible to retrieve the query log until a specified point in time in the past. This would limit the amount of network information flow during the sampling process, making distributed workload collection phase more scalable. Moreover, this would allow applications to make the adaptive OID system more sensitive to the latest workload changes, specially when the adaptation does not happen for a longer period of time because of the effect of the old queries in the history.

The estimation of the index adaptation cost in the adaptation decision phase of the adaptive OID system is based upon a basic query routing strategy (see Section 5.3.3). Using this basic routing strategy, a lookup is first performed for each zone identifier that a query maps to, on an SFC-based index. Next, the query is directly sent to the peers responsible for those identifiers. Development of an adaptation cost estimation formula for the optimized routing strategy utilized by the basic OID system (see Section 3.7.1), is also a part of the future work.

Bibliography

[AB07] Reaz Ahmed and Raouf Boutaba. Distributed Pattern Matching: A Key to Flexible and Efficient P2P Search. *IEEE Journal on Selected Areas in Communications*, 25:73–83, 2007.

[AC08] William Acosta and Surendar Chandra. Exploiting the Properties of Query Workload and File Name Distributions to Improve P2P Synopsis-based Searches. In *Proceedings of the 27th International Conference on Computer Communications (INFOCOM'08)*, pages 2467–2475. IEEE, 2008.

[AR04] Filipe Araújo and Luís Rodrigues. GeoPeer: A Location-Aware Peer-to-Peer System. In *Proceedings of the 3rd International Symposium on Network Computing and Applications (NCA'04)*, pages 39–46. IEEE, 2004.

[ARR$^+$95] Tetsuo Asano, Desh Ranjan, Thomas Roos, Emo Welzl, and Peter Widmayer. Space Filling Curves and Their Use in the Design of Geometric Data Structures. In *Proceedings of the 2nd Latin American Symposium on Theoretical Informatics (LATIN'95)*, pages 36–48. Springer, 1995.

[AX02] Artur Andrzejak and Zhichen Xu. Scalable, Efficient Range Queries for Grid Information Services. In *Proceedings of the 2nd International Conference on Peer-to-Peer Computing (P2P'02)*, pages 33–40. IEEE, 2002.

[BA07] Nabhendra Bisnik and Alhussein A. Abouzeid. Optimizing Random Walk Search Algorithms in P2P Networks. *Computer Networks*, 51:1499–1514, 2007.

[BAS04] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proceedings of the Symposium on Communications Architectures and Protocols (SIGCOMM'04)*, pages 353–366. ACM, 2004.

[BC05] Nicolas Bruno and Surajit Chaudhuri. Automatic physical database tuning:

a relaxation-based approach. In *Proceedings of the International Conference on Management of Data (SIGMOD'05)*, pages 227–238. ACM, 2005.

[Blo70] Burton H. Bloom. Space/time Trade-offs in Hash Coding with Allowable Errors. *Communications*, 13:422–426, 1970.

[CCR05] Miguel Castro, Manuel Costa, and Antony Rowstron. Debunking Some Myths About Structured and Unstructured Overlays. In *Proceedings of the 2nd International Symposium on Networked Systems Design & Implementation (NSDI'05)*, pages 85–98. USENIX Association, 2005.

[CDN04] Surajit Chaudhuri, Mayur Datar, and Vivek Narasayya. Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution. *IEEE Transactions on Knowledge and Data Engineering*, 16:1313–1323, 2004.

[CFCS03] Min Cai, Martin Frank, Jinbo Chen, and Pedro Szekely. MAAN: A Multi-Attribute Addressable Network for Grid Information Services. In *Proceedings of the 4th International Workshop on Grid Computing (GRID'03)*, pages 184–191. IEEE, 2003.

[CGM02] Arturo Crespo and Hector Garcia-Molina. Routing Indices for Peer-to-Peer Systems. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 23–32. IEEE, 2002.

[CLRS03] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms.* MIT Press, 2003.

[CN97] Surajit Chaudhuri and Vivek R. Narasayya. An Efficient Cost-Driven Index Selection Tool for Microsoft SQL Server. In *Proceedings of the 23rd International Conference on Very Large Databases (VLDB'97)*, pages 146–155. Morgan Kaufmann Publishers Inc., 1997.

[CN99] Surajit Chaudhuri and Vivek R. Narasayya. Index Merging. In *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*, page 296. IEEE, 1999.

[CRB$^+$03] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. Making gnutella-like p2p systems scalable. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'03)*, pages 407–418. ACM, 2003.

[CRR$^+$05] Yatin Chawathe, Sriram Ramabhadran, Sylvia Ratnasamy, Anthony LaMarca, Scott Shenker, and Joseph Hellerstein. A Case Study in Building Layered DHT Applications. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'05)*, pages 97–108. ACM, 2005.

[D̈10] Frank Dürr. Concepts of Peer-to-Peer Systems. In *Lecture Notes on Concepts of Peer-to-Peer Systems.* Universität Stuttgart, 2010.

[DFZ+09] Ze Deng, Dan Feng, Ke Zhou, Zhan Shi, and Chao Luo. Range Query Using Learning-Aware RPS in DHT-Based Peer-to-Peer Networks. In *Proceedings of the 9th International Symposium on Cluster Computing and the Grid (CCGRID'09)*, pages 180–187. IEEE, 2009.

[DHJ+05] Anwitaman Datta, Manfred Hauswirth, Renault John, Roman Schmidt, and Karl Aberer. Range Queries in Trie-Structured Overlays. In *Proceeding of the 5th International Conference on P2P Computing (P2P'05)*, pages 57–66. IEEE, 2005.

[DTK08] Katerina Doka, Dimitrios Tsoumakos, and Nectarios Koziris. HiPPIS: An Online P2P System for Efficient Lookups on d-dimensional Hierarchies. In *Proceeding of the 10th Workshop on Web information and Data Management (WIDM'08)*, pages 63–70. ACM, 2008.

[EAABH03] Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi. Efficient Broadcast in Structured P2P Networks. In *Peer-to-Peer Systems II*, volume 2735/2003, pages 304 – 314. Springer, 2003.

[Ear08] NASA Visible Earth. *Earth's City Lights.* `http://visibleearth.nasa.gov/`, 2008.

[fre10] *The Free Haven Project.* `http://www.freehaven.net/`, 2010.

[Fur97] Carlos A. Furuti. *Polyhedral Map Projections.* `http://www.progonos.com/furuti/MapProj/Normal/ProjPoly/projPoly.html`, 1996-97.

[GMUW08] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book.* Prentice Hall Press, 2008.

[gnu10] *The Gnutella Protocol.* `http://rfc-gnutella.sourceforge.net/`, 2010.

[GST07] Adam Shaked Gish, Yuval Shavitt, and Tomer Tankel. Geographical Statistics and Characteristics of P2P query strings. In *Proceedings of 6rd International Workshop on P2P Systems (IPTPS'07)*, 2007.

[GYGM04] Prasanna Ganesan, Beverly Yang, and Hector Garcia-Molina. One Torus to Rule Them All: Multi-dimensional Queries in P2P Systems. In *Proceedings of the 7th International Workshop on the Web and Databases (WebDB'04)*, pages 19–24. ACM, 2004.

[Hil91] David Hilbert. Über die stetige Abbildung einer Linie auf ein Flächenstück. In *Mathematische Annalen*, 1891.

[HT03] Aaron Harwood and Egemen Tanin. Hashing Spatial Content over Peer-to-Peer Networks. In *Proceeding of the Australian Telecommunications, Networks and Applications Conference (ATNAC'03)*, pages 1–5. ATNAC'03, 2003.

[HZ10] Kun Huang and Dafang Zhang. DHT-based Lightweight Broadcast Algorithms in Large-scale Computing Infrastructures. *Future Generation Computer Systems*, 26(3):291–303, 2010.

[IG01] John J. Bartholdi III and Paul Goldsman. Continuous indexing of hierarchical subdivisions of the globe. *International Journal of Geographical Information Science*, 15(6):489–522, 2001.

[kaz10] *Kazaa*. `http://www.kazaa.com/`, 2010.

[KGZY02] Vana Kalogeraki, Dimitrios Gunopulos, and D. Zeinalipour-Yazti. A Local Search Mechanism for Peer-to-Peer Networks. In *Proceedings of the 11th International Conference on Information and Knowledge Management (CIKMss'02)*, pages 300–307. ACM, 2002.

[KLL04] Hye-Young Kang, Bog-Ja Lim, and Ki-Joune Li. P2P Spatial Query Processing by Delaunay Triangulation. In *Proceedings of the 4th International Workshop on Web and Wireless Geographical Information Systems (W2GIS'04)*, volume 3428/2005, pages 136–150. Springer, 2004.

[KLS07] Aleksandra Kovacevic, Nicolas Liebau, and Ralf Steinmetz. Globase.KOM - A P2P Overlay for Fully Retrievable Location-based Search. In *Proceedings of the 7th International Conference on Peer-to-Peer Computing (P2P'07)*, pages 87–96. IEEE, 2007.

[KLVW04] Alexander Klemm, Christoph Lindemann, Mary K. Vernon, and Oliver P. Waldhorst. Characterizing the Query Behavior in Peer-to-Peer File Sharing Systems. In *Proceeding of the 4th Internation Conference on Internet Measurement (IMC'04)*, pages 55–67. ACM, 2004.

[KPPT05] Georgia Koloniari, Yannis Petrakis, Evaggelia Pitoura, and Thodoris Tsotsos. Query Workload-Aware Overlay Construction Using Histograms. In *Proceedings of the 14th International Conference on Information and Knowledge Management (CIKM'05)*, pages 640–647. ACM, 2005.

[KRRS04] Brad Karp, Sylvia Ratnasamy, Sean Rhea, and Scott Shenker. Spurring Adoption of DHTs with OpenHash, a Public DHT Service. In *Proceedings of the 3rd International Workshop on P2P Systems (IPTPS'04)*. Springer, 2004.

[KW06] Mirko Knoll and Torben Weis. Optimizing Locality for Self-organizing Context-Based Systems. In *Proceedings of the 1st International Workshop on Self-Organizing Systems (IWSOS'06)*, volume 4124/2006, pages 62–73. Springer, 2006.

[Law00] Jonathan K. Lawder. Calculation of Mappings between One and n-dimensional Values Using the Hilbert Space-Filling Curve. Technical Report JL1/00, Birkbeck College, University of London, 2000.

[LCC$^+$02] Qin Lv, Pei Cao, Edith Cohen, Kai Li, and Scott Shenker. Search and Replication in Unstructured Peer-to-Peer Networks. In *Proceedings of 16th International Conference on Supercomputing (ICS'02)*, pages 84–95. ACM, 2002.

[LK00] Jonathan K. Lawder and Peter J. H. King. Using Space-Filling Curves for Multi-dimensional Indexing. In *Proceedings of the 17th British National Conferenc on Databases (BNCOD'00)*, pages 20–35. Springer, 2000.

[LK06] Karsten Loesing and Sven Kaffille. *Open Chord*. `http://sourceforge.net/projects/open-chord/`, 2006.

[LT92] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems*. Academic Press Ltd., 1st edition, 1992.

[LW06] Xiuqi Li and Jie Wu. Improve Searching by Reinforcement Learning in Unstructured P2Ps. In *Proceedings of the 26th International Conference on Distributed Computing Systems Workshops (ICDCSW'06)*, page 75. IEEE, 2006.

[MAK03] Mohamed F. Mokbel, Walid G. Aref, and Ibrahim Kamel. Analysis of Multi-Dimensional Space-Filling Curves. *Geoinformatica*, 7(3):179–209, 2003.

[MBR03] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed Hashing in a Small World. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems (USITS'03)*, page 10. USENIX Association, 2003.

[MDR10] Faraz Memon, Frank Dürr, and Kurt Rothermel. Index Recommendation Tool for Optimized Information Discovery Over Distributed Hash Tables. In *Proceedings of 35th International Conference on Local Computer Networks (LCN'10)*, pages 104–111. IEEE, 2010.

[MJFS01] Bongki Moon, H. v. Jagadish, Christos Faloutsos, and Joel H. Saltz. Analysis of the Clustering Properties of the Hilbert Space-Filling Curve. *IEEE Transactions on Knowledge and Data Engineering*, 13(1):124–141, 2001.

[MK02] Daniel A. Menascé and Lavanya Kanchanapalli. Probabilistic scalable P2P resource location services. *SIGMETRICS Performance Evaluation Review*, 30:48–58, 2002.

[MN04] Chip Martel and Van Nguyen. Analyzing Kleinberg's (and Other) Small-world Models. In *Proceedings of the 23rd International Symposium on Principles of Distributed Computing (PODC'04)*, pages 179–188. ACM, 2004.

[MNR02] Dahlia Malkhi, Moni Naor, and David Ratajczak. Viceroy: A Scalable and Dynamic Emulation of the Butterfly. In *Proceedings of the 21st Annual Symposium on Principles of Distributed Computing (PODC'02)*, pages 183–192. ACM, 2002.

[MRPM08] Elena Meshkova, Janne Riihijärvi, Marina Petrova, and Petri Mähönen. A Survey on Resource Discovery Mechanisms, Peer-to-Peer and Service Discovery Frameworks. *Computer Networks*, 52:2097–2128, 2008.

[MTD$^+$08] Faraz Memon, Daniel Tiebler, Frank Dürr, Kurt Rothermel, Marco Tomsu, and Peter Domschitz. OID: Optimized Information Discovery using Space Filling Curves in P2P Overlay Networks. In *Proceedings of 14th International Conference on Parallel and Distributed Systems (ICPADS'08)*, pages 311–319. IEEE, 2008.

[MTD$^+$09] Faraz Memon, Daniel Tiebler, Frank Dürr, Kurt Rothermel, Marco Tomsu, and Peter Domschitz. Scalable Spatial Information Discovery over DHTs. In *International Conference on Communication System Software and Middleware (COMSWARE'09)*, pages 1–12. ACM, 2009.

[MTDR10] Faraz Memon, Daniel Tiebler, Frank Dürr, and Kurt Rothermel. Optimized Information Discovery using Self-adapting Indices over Distributed Hash Tables. In *Proceedings of 29th International Performance Computing and Communications Conference (IPCCC'10)*, pages 105–113. IEEE, 2010.

[ND08] Jeonghun Noh and Sachin Deshpande. Pseudo-DHT: Distributed Search Algorithm for P2P Video Streaming. In *Proceedings of the 10th International Symposium on Multimedia (ISM'08)*, pages 348–355. IEEE, 2008.

[NYF08] Linh Thai Nguyen, Wai Gen Yee, and Ophir Frieder. Query Workload Driven Summarization for P2P Query Routing. In *Proceedings of the 8th International Conference on Peer-to-Peer Computing (P2P'08)*, pages 63–72. IEEE, 2008.

[OM84] J. A. Orenstein and T. H. Merrett. A Class of Data Structures for Associative Searching. In *Proceedings of the 3rd SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS'84)*, pages 181–190. ACM, 1984.

[oSN] National Institute of Standards and Technology (NIST). *Secure Hash Algorithm*. `http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf`.

[Pea90] Giuseppe Peano. Sur une courbe, qui remplit toute une aire plane (On a Curve Which Completely Fills a Planar Region). In *Mathematishe Annalen*, 1890.

[pee] *PeerSim: A Peer-to-Peer Simulator*. `http://peersim.sourceforge.net/`.

[PRR97] C. Greg Plaxton, Rajmohan Rajaraman, and Andréa W. Richa. Accessing nearby copies of replicated objects in a distributed environment. In *Proceedings of the 9th Annual Symposium on Parallel Algorithms and Architectures (SPAA'97)*, pages 311–320. ACM, 1997.

[Pug90] William Pugh. Skip Lists: A Probabilistic Alternative to Balanced Trees. *Communications*, 33:668–676, 1990.

[QNS02] Changtao Qu, Wolfgang Nejdl, and Holger Schinzel. Integrating Schema-specific Native XML Repositories into a RDF-based e-learning P2P Network. In *Proceedings of International Conference on Dublin Core and Metadata Applications: Metadata for E-Communities: Supporting Diversity and Convergence*, pages 81–89. Dublin Core Metadata Initiative, 2002.

[RD01] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location and Routing for Large-scale Peer-to-Peer Systems. In *Proceedings of IFIP/ACM International Conference on Distributed Systems Platforms*, pages 329–350. ACM, 2001.

[RFH$^+$01] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A Scalable Content-addressable Network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 161–172. ACM, 2001.

[RHRS04] Sriram Ramabhadran, Joseph Hellerstein, Sylvia Ratnasamy, and Scott Shenker. Prefix Hash Tree - An Indexing Data Structure over Distributed Hash Tables. In *Proceedings of the 23rd Symposium on Principles of Distributed Computing (PODC'04)*. Springer, 2004.

[Rip01] Matei Ripeanu. Peer-to-Peer Architecture Case Study: Gnutella Network. In *Proceedings of the 1st International Conference on Peer-to-Peer Computing (P2P'01)*, pages 99–100. IEEE, 2001.

[RLS$^+$03] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica. Load Balancing in Structured P2P Systems. In *Peer-to-Peer Systems II*, volume 2735 of *Lecture Notes in Computer Science*, pages 68–79. Springer, 2003.

[SA06] Gleb Skobeltsyn and Karl Aberer. Distributed Cache Table: Efficient Query-driven Processing of Multi-term Queries in P2P Networks. In *Proceedings of the International Workshop on Information Retrieval in Peer-to-Peer Networks (P2PIR'06)*, pages 33–40. ACM, 2006.

[SAAA05] Ozgur D. Sahin, Shyam Antony, Divyakant Agrawal, and Amr El Abbadi. PRoBe: Multi-dimensional Range Queries in P2P Networks. In *Proceed-*

ings of the 6th International Conference on Web Information Systems Engineering (WISE'05), pages 332–346. Springer, 2005.

[Sag94] Hans Sagan. *Space-Filling Curves*. Springer, 1994.

[SBR04] Nima Sarshar, P. Oscar Boykin, and Vwani P. Roychowdhury. Percolation Search in Power Law Networks: Making Unstructured Peer-to-Peer Networks Scalable. In *Proceedings of the 4th International Conference on Peer-to-Peer Computing (P2P'04)*, pages 2–9. IEEE, 2004.

[SGG03] Stefan Saroiu, Krishna P. Gummadi, and Steven D. Gribble. Measuring and Analyzing the Characteristics of Napster and Gnutella Hosts. *Multimedia Systems*, 9(2):170–184, 2003.

[SMK$^+$01] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM'01)*, pages 149–160. ACM, 2001.

[SOTZ05] Yanfeng Shu, Beng Chin Ooi, Kian-Lee Tan, and Aoying Zhou. Supporting Multi-Dimensional Range Queries in Peer-to-Peer Systems. In *Proceedings of the 5th International Conference on Peer-to-Peer Computing (P2P'05)*, pages 173–180. IEEE, 2005.

[SP03] Cristina Schmidt and Manish Parashar. Flexible Information Discovery in Decentralized Distributed Systems. In *Proceedings of the 12th International Symposium on High Performance Distributed Computing (HPDC'03)*, pages 226–235. IEEE, 2003.

[SP04] Cristina Schmidt and Manish Parashar. Analyzing the Search Characteristics of Space Filling Curve-based Indexing with the Squid P2P Data Discovery System. Technical Report TR-276, CAIP, Rutgers University, 2004.

[Sri01] Kunwadee Sripanidkulchai. The Popularity of Gnutella Queries and its Implications on Scalability, 2001.

[SW05] Ralf Steinmetz and Klaus Wehrle. *Peer-to-Peer Systems and Applications*. Springer, 2005.

[SX07] Haiying Shen and Cheng-Zhong Xu. Locality-Aware and Churn-Resilient Load-Balancing Algorithms in Structured Peer-to-Peer Networks. *IEEE Transactions on Parallel Distributed Systems*, 18(6):849–862, 2007.

[SXC06] Haiying Shen, Cheng-Zhong Xu, and Guihai Chen. Cycloid: A Constant-degree and Lookup-efficient P2P Overlay Network. *Performance Evaluation*, 63:195–216, 2006.

[THS$^+$04] Egemen Tanin, Aaron Harwood, Hanan Samet, Sarana Nutanong, and Minh Tri Truong. A serverless 3D World. In *Proceedings of the 12th International workshop on Geographic information systems (GIS'04)*, pages 157–165. ACM, 2004.

[Tie08] Daniel Tiebler. Optimierung multidimensionaler Bereichsanfragen mittels raumfüllender Kurven in Peer-to-Peer-Netzen. Diploma thesis, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, August 2008.

[TP03] Peter Triantafillou and Theoni Pitoura. Towards a Unifying Framework for Complex Query Processing over Structured Peer-to-Peer Data Networks. In *Proceedings of International Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P'03)*, pages 169–183. Springer, 2003.

[tpc] *Transaction Processing Performance Council.* `http://www.tpc.org/`.

[TR03] Dimitrios Tsoumakos and Nick Roussopoulos. Adaptive Probabilistic Search for Peer-to-Peer Networks. In *Proceedings of the 3rd International Conference on Peer-to-Peer Computing (P2P'03)*, pages 102–109. IEEE, 2003.

[TR06] Dimitrios Tsoumakos and Nick Roussopoulos. Analysis and Comparison of P2P Search Methods. In *Proceedings of the 1st International Conference on Scalable Information Systems (InfoScale'06)*. ACM, 2006.

[TS10] Sabu M. Thampi and K. Chandra Sekaran. An Enhanced Search Technique for Managing Partial Coverage and Free Riding in P2P Networks. *Computing Research Repository*, abs/1006.1017, 2010.

[VZZ$^+$00] Gary Valentin, Michael Zuliani, Daniel C. Zilio, Guy Lohman, and Alan Skelley. DB2 Advisor: An Optimizer Smart Enough to Recommend its own Indexes. *Proceedings of 13th International Conference on Data Engineering (ICDE'00)*, 0:101–110, 2000.

[WZK05] Haojun Wang, Roger Zimmermann, and Wei-Shinn Ku. ASPEN: An Adaptive Spatial Peer-to-Peer Network. In *Proceedings of the 13th International Workshop on Geographic Information Systems (GIS'05)*, pages 230–239. ACM, 2005.

[YGM02] Beverly Yang and Hector Garcia-Molina. Improving Search in Peer-to-Peer Networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems (ICDCS'02)*, pages 5–14. IEEE, 2002.

[YLY07] Fuyong Yuan, Jian Liu, and Chunxia Yin. A Scalable Search Algorithm on Unstructured P2P Networks. In *Proceedings of the 8th International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed (SNPD'07)*, pages 199–204. IEEE, 2007.

[YS07] Liu J. Yin C. Liang S. Yuan, F. and N. Shen. A Novel Search Algorithm Utilizing High Degree Nodes. In *Proceedings of the 2nd International Conference on Communications and Networking in China (CHINACOM'07)*, pages 44–48. IEEE, 2007.

[ZCS05] Hai Zhuge, Xue Chen, and Xiaoping Sun. Preferential Walk: Towards Efficient and Scalable Search in Unstructured Peer-to-Peer Networks. In *Special Interest Tracks and Posters of the 14th international Conference on World Wide Web (WWW'05)*, pages 882–883. ACM, 2005.

[ZH06] Yingwu Zhu and Yiming Hu. Enhancing Search Performance on Gnutella-Like P2P Systems. *IEEE Transactions on Parallel Distributed Systems*, 17:1482–1495, 2006.

[ZHS$^+$04] B.Y. Zhao, Ling Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry: a resilient global-scale overlay for service deployment. 22:41–53, 2004.

[ZLXN03] Zhenyun Zhuang, Yunhao Liu, Li Xiao, and Lionel M. Ni. Hybrid Periodical Flooding in Unstructured Peer-to-Peer Networks. *International Conference on Parallel Processing (ICPP'03)*, 0:171, 2003.

[ZRL$^+$04] Daniel C. Zilio, Jun Rao, Sam Lightstone, Guy Lohman, Adam Storm, Christian Garcia-Arellano, and Scott Fadden. DB2 Design Advisor: Integrated Automatic Physical Database Design. In *Proceedings of the 13th International Conference on Very Large Data Bases (VLDB'04)*, pages 1087–1097. VLDB Endowment, 2004.

[ZZSL07] Haoxiang Zhang, Lin Zhang, Xiuming Shan, and V.O.K. Li. Probabilistic Search in P2P Networks with High Node Degree Variation. In *Proceedings of International Conference on Communications (ICC'07)*, pages 1710 –1715. IEEE, 2007.