7. Peer-to-Peer Data Management

Theo Härder
www.haerder.de

Motivation

Searching in peer-to-peer systems (routing indexes)

Scalable distributed data structures

using Chord, CAN, P-Grid

Distributed hash tables for structured data

Main reference:
Gunter Saake, Andreas Heuer, Kai-Uwe Sattler : Datenbanken - Implementierungstechniken, 2. Auflage, mitp-Verlag, 2005, Kap. 13.3

P2P: Network without Centralized Control

- Three main requirements of future Internet-based applications
  - scalability
    - vast demand of bandwidth, storage capacity. and processing power
    - caused by a large number of users
  - security and reliability
    - are core requirements for the availability of strategically important and security-sensitive services
    - especially, in the face of distributed denial-of-service attacks on centralized systems
    - anonymity and resistance of censorship are today of growing importance
  - flexibility and quality of service (QoS)
    - must enable quick and easy integration of new services
    - for example, services enabling group communication and mobility

  → client-server approaches fail!
- Definition
  - a P2P system is a self-organizing system of equal, autonomous entities (peers) which aims at the shared usage of distributed resources in a networked environment avoiding central services

→ a system with completely decentralized self-organization and resource usage

P2P: Network without Centralized Control (2)

- Decentralized resource usage
  - resources are used in a manner as equally distributed as possible and are located at the edges of the network, close to the peers
  - within a set of peers, each utilizes the resources provided by the other peers
  - peers are interconnected through a (globally distributed) network
  - a peer's Internet address typically changes so the peer is not always reachable at the same address (transient connectivity)
  - peers may dynamically connect or disconnect (shut down over longer periods of time)
- Decentralized self-organization
  - in order to utilize shared resources, peers directly interact with each other
  - they directly access and exchange the shared resources without a centralized service – performance considerations may lead to centralized elements (hybrid P2P)
  - peers can interact both as clients and servers (servents) or act as routers
  - peers are equal partners with symmetric functionality (fully autonomous regarding their resources)
  - ideally, resources can be located without any central entity or service

P2P: Network without Centralized Control (3)

- Paradigm shift from
  - coordination to cooperation
  - centralization to decentralization
  - control to incentives

  → use of (network / foreign peer) services paid by "altruistic" behavior
- Loose coupling of many autonomous computers
  - each node only knows some of its "direct" neighbors (no global knowledge)
- Applications:
  - large-scale computation (SETI@home, etc.)
  - file sharing (Napster, Gnutella, KaZaA, etc.)
  - publish-subscribe (Blogs, marketplaces, etc.)
  - collaborative work (games, etc.)
  - collaborative Web search and data mining (P2P information retrieval)
- Goals:
  - make systems ultra-scalable and self-organizing
  - make systems manageable and less susceptible to attacks
  - break information monopolies, exploit small-world phenomenon[1]

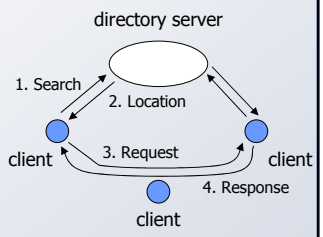
1) The **small world phenomenon** (also known as the **small world effect**) is the hypothesis that everyone in the world can be reached through a short chain of social acquaintances. The concept gave rise to the famous phrase six degrees of separation after a 1967 **small world experiment** by social psychologist Stanley Milgram which suggested that two random US citizens were connected on average by a chain of six acquaintances.

Application: File Sharing

- **Idea**
  - exchange of locally stored files (e.g. music files) among user groups formed in an ad-hoc manner
  - (illegal) file sharing leveraged P2P technology
  - system support for search operations, caching, directory services, etc.
  - goals: anonymity, privacy
- **Advantages of distributed storage**
  - no "single point of failure" / bottleneck
  - load distribution (here storage requirement) resp. cost (example: Napster: 7 TB per day!)
- **Multiplicity of P2P projects, services, systems, ...**
  - examples: Napster, Gnutella, Freenet, ...
    - distributed storage / administration of data/files (e. g. MP3 files)
    - search via central server (Napster) or distributed (Gnutella)
  - systems: Napster, Gnutella, MojoNation, Freenet, ...

Example: Napster

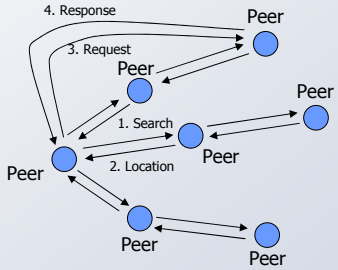
- developed by Shawn Fanning in 1999, MP3 file sharing



- central directory server (index of files offered)
  - connection of peers (clients)
  - name space independent of DNS
  - registration of files offered at directory server (MD5 hash, file infos ID3-Tags, communication bandwidth, ...)
  - search via attributes (title, album, interpreter, ...) in central index and result to client
  - selection in client and additional request for IP address & port
  - direct connection of clients for download
- additional features: chat, personalization, local file management
- comment
  - not existing anymore in the original version!!!

Example: Gnutella

- originally developed as "America Online" project (Nullsoft), stop because of potential Copyright violation
- reengineering of the protocol (Brian Maryland) → Open-Source project



- base principle
  - decentralized P2P file sharing + search
  - each Gnutella application (SERVer + cliENT) is client as well as server
  - different servents (LimeWire, BearShare, ...)
  - participation in the Gnutella network requires acquaintance of a peer (IP address), exchange of peer lists
- problems
  - flooding: flow of messages always via all known peers / direct download → scalability

Trends in DBMSs

Motivation
Routing indexes
Chord
CAN
P-Grid
DHT for structured data
Summary

Problems of Gnutella

messages: unique identification per random number
typical transmission: N=<5, TTL=7

reachability (balanced network)

hops

	TTL=1	2	3	...	7	8
N=2	2	4	6	...	...	...
3	3	9	21	...	...	...
4	4	16	52	...	4372	...
...	...	...	...	...	...	...
8	...	...	...	...	...	7686400

connections per peer

bandwidth generation (data packets ca. 83 bytes)

	TTL=1	2	3	...	7	8
N=2	166	332	498	...	1162	...
3	249	747	1743	...	31623	...
4	...	...	...	...	362876	...
...	...	...	...	...	...	...
8	...	...	...	...	...	637971200

10-60% of bandwidth for pings!

ping is a computer program which can check, whether a certain host is reachable in an IP-network and with which response time it can react

Problems of Gnutella (2)

- **Fragmentation**
  - limitation of packet transfer because of restricted bandwidth and large number of messages
  - with increasing number of users: fragmentation in isolated subnets
  - reduction of the number of reachable peers to 300-500
- **Free riding**
  - large number of users and anonymity
  - user do not contribute anymore (offering of files), but only "consume"
  - system vulnerability grows, because a few systems "play" the role of central servers
  - survey of Gnutella (Xerox, August 2000)
  - 66% of peers: no files, 73%: ← 10 files
  - 1% of peers: 37% of files, 20%: 98% of files
  - 63% of peers do not respond to requests
  - 1% of peers: 47% of replies, 25%: 98% replies

Freenet

- **Open source: www.freenetproject.org**
  - distributed decentralized system, originally developed at the Univ. Edinburgh
  - sharing of storage space instead of files
- **Base principle**
  - peer makes public storage space available, to be used by everybody
  - files are identified by object keys; kept in global storage
- **Administration of peers**
  - management of information w.r.t. content (key, node of origin, access rights, ...)
  - surveillance of utilization of storage space
  - LRU caching of contents, longer storage of metadata → enables access of the original version of deleted data
- **Access**
  - determination of object key
  - request to local node (key, hops-to-live)
  - if unsuccessful: lookup in routing table for "neighbor" key → transmission to corresponding node resp. flooding
- **Maintenance of routing tables**
  - during processing of requests
  - by "self-registration" of nodes

Freenet is free software which lets you publish and obtain information on the Internet without fear of censorship. To achieve this freedom, the network is entirely decentralized and publishers and consumers of information are anonymous. Without anonymity there can never be true freedom of speech, and without decentralization the network will be vulnerable to attack.

Search in P2P Systems

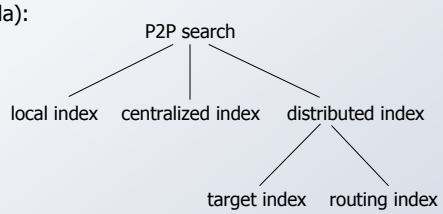
- **Key property: scalability**
  - Napster, Gnutella, etc.: scalable file transfer, but not search / indexing
- **Indexing as classification aspect**
  - central, local, distributed
  - non-forwarding (goal), forwarding (direction)
  - distributed index structure, semantic routing

  → possibly super-peer architecture (hierarchical approach, structured routing)
- **Query classes**
  - lookup and range queries
  - top-k & nearest neighbor
  - join queries, aggregations
  - miscellaneous: recursion, IR queries, continuous queries, ...

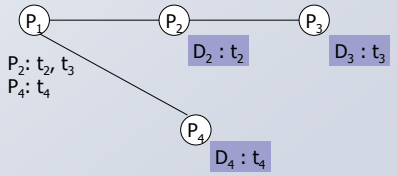
  → only approximate results!

Routing Indexes

- **In the past**
  - local (or missing) index (Gnutella): flooding → high network load
  - centralized index (Napster): bottleneck, vulnerable, anonymity not guaranteed
  - distributed index (Freenet)



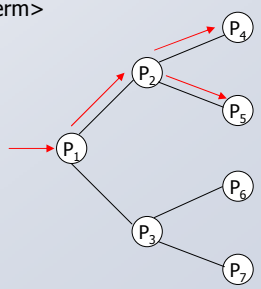
- **Idea**
  - refinement of distributed indexes
  - reduction of network load through directed forwarding of messages
- **Principle**
  - indexing of "direction" (route) to the documents relevant to specific queries
  - route selection based on index entries
  - index size ~ number of neighbor nodes (target index: size ~ number of documents)



Routing Indexes (2)

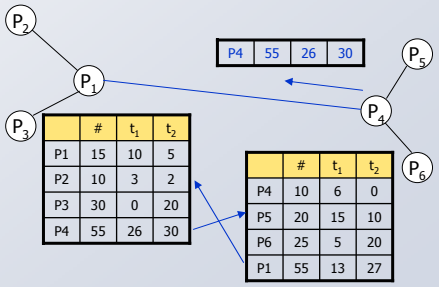
- **Query model**
  - query evaluation on local DB; delivering of results, if present
  - if stop criterion not satisfied (TTL value)
    - forward query to one or several neighbors (parallel or sequential)
    - sequence and number of neighbors dependent on the quality q w.r.t. the query Q (set of terms t)
  - quality criterion: number of documents relevant for the terms t ∈ Q
- **Structure**
  - index per peer
  - index entry RI: <neighbor → NoDocuments, NoDocsPerTerm>
- **Example for peer P1**

neighbor	no. of documents	no. of documents having term	
		t1	t2
P2	50	30	5
P3	20	0	15



Routing Indexes (3)

- **Usage**
  - quality of path via neighbor n for query Q: $q(n,Q) = A_n \cdot \prod_{t \in Q} \frac{RI(n,t)}{A_n}$
    - $RI(n,t)$: no. of documents having term t via neighbor n
    - A_n: no. of documents for n
  - problem: no consideration of costs (e.g. caused by many hops)
    - limitation of #hops (hop-count routing index)
- **Creation**
  - done when a connection is created between two peers P_i and P_j
    - in each case, aggregation of vectors of RI
    - exchange of vectors
    - in each case, forwarding to neighbor N_i exclusive of index entries of N_i



→ maintenance of indexes?

Scalable Distributed Data Structures

- **Key requirement for P2P systems: scalability**
  - in the kernel: indexing scheme: *file name → storage location*
  - but: common properties of routing indexes
    - "fixed schema" – data description wired in the application (maintenance!)
    - simple keyword-based queries
- **Structured P2P systems**
  - scalable indexing mechanism → data structure
    (SDDS: Scalable Distributed Data Structures)
  - distributed content-based query routing
  - Principle: management of (key, value) pairs

  → **distributed hash tables (DHT):**
  logical keys allow messages to be transferred to the peers responsible for the keys
- **Design objectives**
  - load balancing: equi-distribution of keys across all nodes
  - decentralization: only equal nodes, no specialized nodes
  - availability:
    adjustment of structure when nodes join or leave the network, or crash
  - flexible naming scheme: no restrictions w.r.t. key structure

DHT: Data Organization

- **Characteristics**
  - each peer only knows a small number of neighbors
    (address, range of keys)
  - reachability of peers in a network of n peers: O(log n) hops
  - each peer can independently decide how to forward messages
    (greedy strategy for the shortest path)
  - entire system should be robust against node crashes; forwarding of messages should be guaranteed even in case of a crash

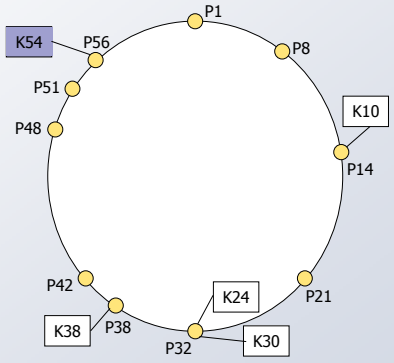
  → DHT-based systems (Chord, CAN, P-Grid) essentially differ w.r.t. their topology
- Conceptual API is similar
  - operations (executable by each peer)
    - *put(K, V)*: storing of the pair *(K, V)*
    - *get(K)* → *V*: determination of value *V*, given *K*
  - basis: *lookup(K)*
    - if *h(K)=P* is not managed by the requesting node
      → forwarding (Routing)

Chord – Distributed Lookup Protocol

- **Objective**
  - mapping of key values onto nodes which manage the related data
  - topology: peers form a ring (identifier circle)
  - hash function (*consistent hashing*, e.g. SHA-1)
    - m-bit identifier (ID) (typically m=160)
    - approximately obtains equi-distribution of keys across all nodes
    - in case of addition / removal of the *N*-th node only *O(1/N)* keys must be displaced
  - node does not need to know all nodes
    - for *N* nodes : *O(log N)* entries
    - only little routing information necessary

Chord (2)

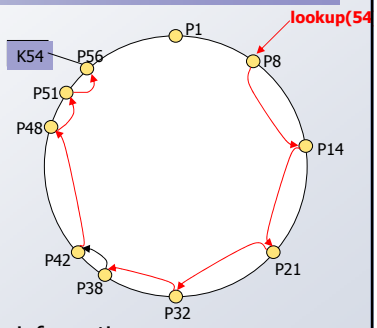
- **Data organization**
  - hash function assigns to each node and each key an *m*-bit identifier ID
  - data structure: *Identifier Circle* (Chord ring)
    - allocation of IDs in a ring modulo 2^m
    - key *k*: is allocated on first node whose ID is equal to the ID of *k* or follows this ID (successor node *successor(k)*)
    - ring with IDs of 0...2^m-1: *successor(k)* is first node after *k* in clockwise direction



Chord: Search

- **Naive approach**
  - each node only knows its successor in the ring
  - query with a given key-ID is passed on along the ring until a match is found

  → node crash?



- **Improved search**
  - routing table with additional routing information (addresses of nodes)
    - each node has *finger table* with max. *m* entries
    - *i*-th entry at node *n*: ID of the first node *s* which is at least 2^{i-1}-th successor of *n*:
      $s = successor(n+2^{i-1})$ with $1 <= i <= m$
      (arithmetic modulo 2^m)
    - *s* is *i*-th *finger* of *n*
  - search:
    determination of highest predecessor for wanted ID

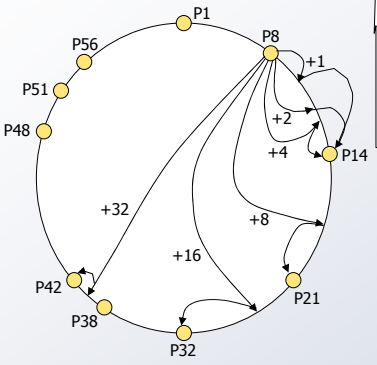
Trends in DBMSs
Motivation
Routing indexes
Chord
CAN
P-Grid
DHT for structured data
Summary

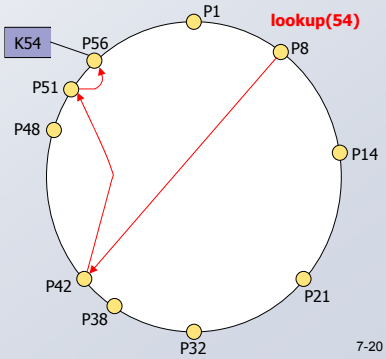
Example: Improved Search using Chord

keys of P8: *range 2 - 8*

finger table

P8+1	P14
P8+2	P14
P8+4	P14
P8+8	P21
P8+16	P32
P8+32	P42

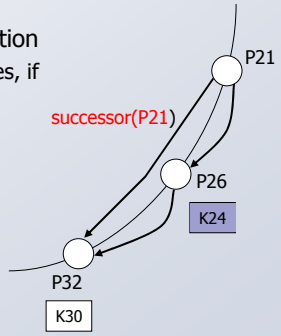




evaluation
- small *finger* table
- cost of search: *O(log N)*

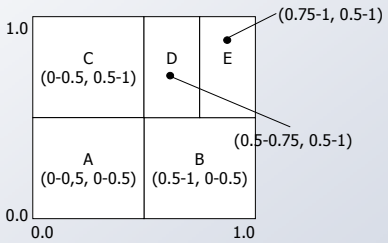
Dynamic Operations

- **Change in the network structure**
  - join: arrival of new nodes in the network
    - determination of the successor
    - update of the *finger* table
    - redistribution of keys
    - but: possibly not all data structures up-to-date so far
  - stabilization protocol – periodic execution
    - checking and update of all finger tables, if necessary
      - determination of the predecessor of the successor
    - centralized service?
- **Example: join**

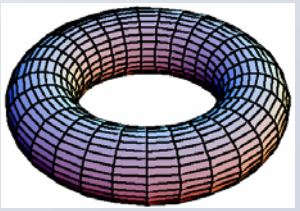


Content Addressable Network (CAN)

- **Chord problem: crash of nodes leads to a ring break**
  - way out: each node has successor list of length *r*
    - in case of failure: contact the next reachable successor
  - approach possible using replication of keys
    - key *k* is inserted in *r* successors
- **CAN**
  - *change of topology:* *d*-dimensional Cartesian coordinate space on a *d*-torus
  - coordinate space is partitioned in such a way that each node administrates an own zone



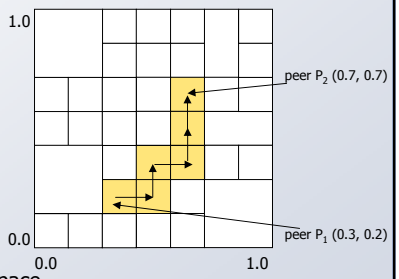
key space for d=2



torus for d=2

CAN: Routing

- **Storage**
  - key *K* is mapped onto point *P* in the coordinate space using a hash function $h(K)=P$ $(= c_1, c_2, \ldots, c_d)$
  - (K, V) is stored on node in zone *Z* which contains *P*
- **Intuitive approach**
  - routing: "straightforward" through the coordinate space
  - per peer: coordinate-routing table
    - for each neighbor: IP address + coordinate zone
    - neighborhood for *d*-dimensional space
      - common boundary: 1 dimension
      - adjacent: d-1 dimensions
    - local neighborhood relationship

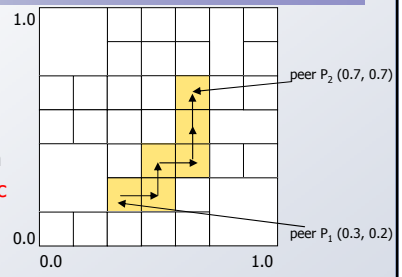


- **Routing**
  - message contains target coordinates
  - greedy forwarding to neighbor next to the target coordinates

Trends in DBMSs
Motivation
Routing indexes
Chord
CAN
P-Grid
DHT for structured data
Summary

CAN: Routing (2)

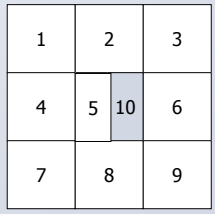
- **Cost estimation**
  - *d*-dimensional space, *n* equally large zones
    - avg. routing path length (hop count): avg. hc = $d/4 * \sqrt[d]{n}$
      → the larger d, the shorter avg. hc
    - each node manages at least $2d$ neighbors
      → #neighbors grows with d



  - scalability: growth of the number of nodes
    - no growth of routing information
    - path length: $O(\sqrt[d]{n})$
  - in addition
    - several routes possible: in case of a neighbor crash, forwarding via "deviation route"

Construction of a CAN

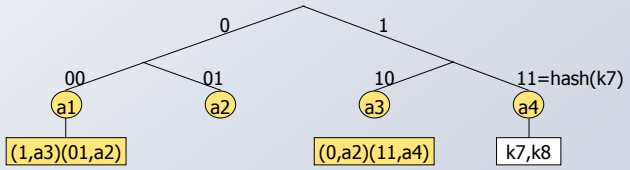
- **Expansion of the CAN**
  - integration of further nodes → new nodes must obtain own zones
    - split of an existing zone and division of data
  - method
    - starting at a (given, arbitrary) peer in the CAN
    - location of a (randomly chosen) peer whose zone should be split (use of CAN routing) & split
    - communicating the split to the neighbors
- **Split of a zone**
  - division of the zone in two halves; one half goes to new peer
  - predetermined order of dimensions for split
  - distribution of (key, value) pairs corresponding to the new zones
- **Update of routing information**
  - new peer: takes over addresses of neighbors from peer of split zone *Z*
  - previous owner of *Z*: update of the new neighbor
  - both: notification of neighbors concerning update
- **Cost**
  - *O(d)* peers affected by updates



5: {2, 10, 8, 4}
10: {5, 2, 6, 8}
2: {5, 10, 3, 8, 1}
. . .

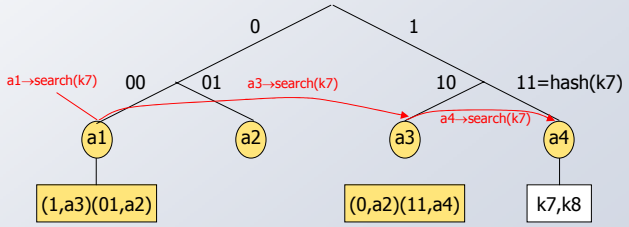
P-Grid: Data Organization

- **Approach**
  - distribution of a virtual binary search tree to several, potentially "unreliable" peers
    - decentralized management without central infrastructure
    - all peers as entry points
    - randomized method for the construction of the access structure
- **Data organization**
  - peer is responsible for interval of keys, i.e., knows all peers having data objects belonging to a key k_{query}
    - data object is stored at peer, if node ID is prefix of the key
  - key *k* corresponds to path in the tree (e.g., k7)
  - for each prefix k_l of length *l*, peer manages references to other peers (only shown for a1 and a3) which
    - have the same prefix of length *l*,
    - however, another value at position *l+1*



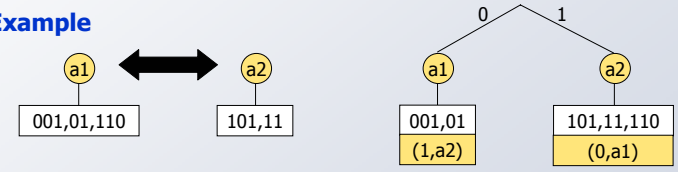
P-Grid: Search

- **Query**
  - routing via responsible peers
  - at next level
    - case 1: peer itself is responsible → further processing
    - case 2: forwarding to responsible peer
      - if multiple possibilities: random selection
  - depth search
- **Search: example**



Construction of a P-Grid

- exclusively local interaction between peers
- **Initial state**
  - each peer only knows own data (all search keys)
- **Meeting of two peers**
  - information exchange and adjustment of both structures
  - split of search spaces into two halves
  - take over of responsibility for one half
  - insertion of a reference to other peer
- **Example**



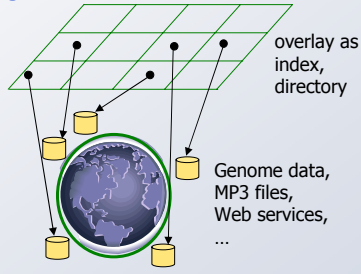
- **Further cases**
  - common prefix of keys of two peers
    - information exchange up to the level of same prefixes
  - keys are related by their prefixes
    - peer having shorter key has to specialize it → key extension
- **Remark**
  - avoidance of over-specialization by fixing of a maximal path length *maxlength* (global knowledge?!)

Comparison of Different Approaches

	Paradigm	kind of search	cost of search (#msgs)
Gnutella	breadth-first search on graph	string comparison	$2 \cdot \sum_{i=0}^{TTL} C \cdot (C-1)^i$
Freenet	depth-first search on graph	equality	O(log n)
Chord	implicit binary search tree	equality	O(log n)
CAN	d-dimensional space	equality	$O\left(d \cdot n^{\frac{1}{d}}\right)$
P-Grid	binary prefix tree	prefix	O(log n)

DHTs for Structured Data

- **Storage of relational data in a CAN**
  - original form of DHTs not suitable
  - fragmentation of relations using appropriate hash functions
  - but: accesses not only via key attribute
  - requires: locality preserving hash functions, temporary rehashing

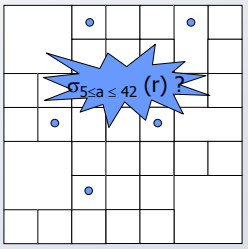


- **Support of SQL queries**
  - lookup operation of CAN are not sufficient for typical DB applications
  - fragmented storage of relations requires parallel realization of query operations

- **Consideration of P2P characteristics (no guarantees)**
  - best-effort techniques
  - no ACID!

Storage of Relational Data

- record $t \in r$, $t = \{a_k, a_1, \ldots, a_n\}$
- hash function $k = h(a_k)$
- problems:
  - records may be unevenly distributed across key space
  - only exact-match queries possible via primary key

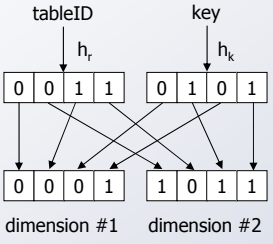


- **Fragmentation scheme: naive approaches**
  - primary key value / resource ID (table ID) as CAN key
  - table ID / key in separate dimensions
  - improvements
    - suitable selection and combination of hash functions: locality preserving hashing, e.g. space-filling curve (Hilbert curve, Z curve, ...)
      → range queries using an efficient multi-cast routing
    - indexing of several attribute
    - generic scheme

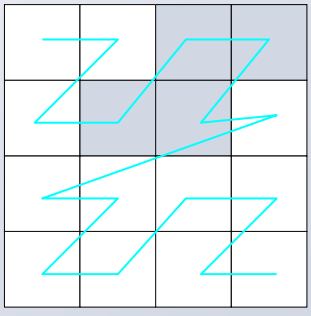
  → exotic approaches?

Fragmentation Scheme

- **Fragmentation via Z curve**
  - goal: locality sensitive fragmentation
  - approach: reverse bit interleaving
    - record $t \in r$, $t = \{a_k, a_1, \ldots, a_n\}$
    - key $k = h_r(r) \circ h_k(a_k)$

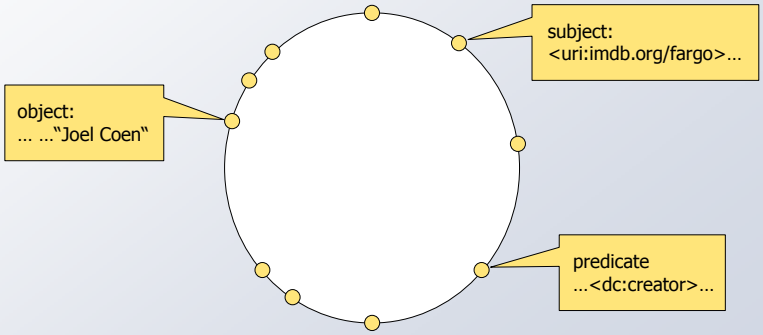


(tableID, key value)



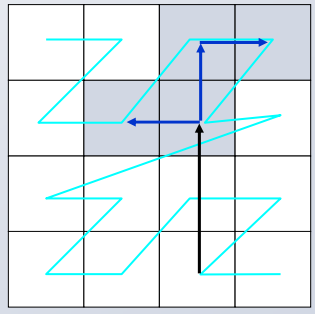
RDF Peers

- Replication in Chord
  - storage of RDF data
  - RDF triple of <subject, predicate, object>
    <uri:imdb.org/fargo><dc:creator> "Joel Coen"
  - approach: threefold insertion of a triple



Query Operators

- **Prerequisite**
  - fragmentation scheme $h_T(T) \cdot h_K(t(A_k))$
  - query routing by DHT own routing
    - stepwise processing of queries
    - "operator shipping":
      message containing query is sent to peers carrying data
  - hash-based implementations
    for selection, join,
    grouping / aggregation



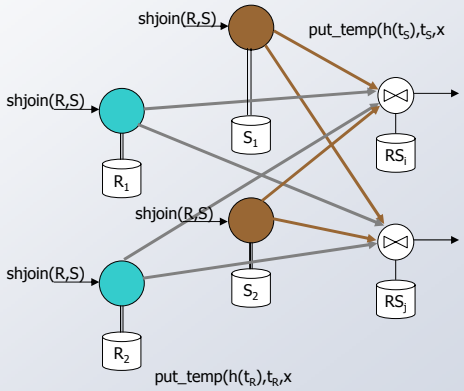
- **API extensions**
  - standard operations: *put*, *get*
  - additionally
    - *put_temp(k, v, t)*: temporary insertion, e.g. for rehasing
    - send_message(z, , m):
    - *multicast*(z_{min}, z_{max}, m): group message m to peers of the z-curve interval z_{min}, z_{max}, e.g. for range queries

Lookup- and Range Queries

- **Lookup queries (exact match)**
  - one-dimensional on key attribute: DHT lookup (get(k))
  - one-dimensional on non-key attribute: scan (flooding)
  - multi-attribute queries
    - preferred attribute
    - special topology (multi-attribute addressable network MAAN)
- **Range queries**
  - on key attribute (when locality-preserving storage): multicast (directed flooding)
  - on non-key attribute: scan (flooding)

Join Queries

- **Symmetric hash join**
  - for peers having records for R or S
    - temporary insertion in table RS
  - peers for RS locally compute join



- **Fetch matches / ship where needed**
  - peers having one of both tables
    - fetch join candidate for each record using the join value (fetch matches)
    - or ship record to peer storing candidate (ship where needed)

Join Queries (2)

- **Semi join**
  - for R- and S-peers, respectively: projection on join attribute and key
  - *symmetric hash join*
  - finishing *fetch-matches join* using complete tables
  - improvement: *Bloom-filter join*
    - Bloom filter on each peer for R or S and insertion in temporary table
- **Grouping / aggregation**
  - naive approach
    - grouping / aggregation as finishing operation locally processed at initiator
  - hash-based grouping
    - temporary insertion of records in "grouping table" using the group-determining value
    - grouping peers: computation of aggregates
  - special aggregation protocols

Trends in DBMSs

Motivation

Routing indexes

Chord

CAN

P-Grid

DHT for structured data

Summary

Similarity-Based Queries

- **Text similarity for attribute values (like)**
  - q-grams: set of all substrings of length q
    Humboldt = { __H, _Hu, Hum, umb, mbo, bol, old, ldt, dt_, t__ }
  - search: q-qrams as key in index
    - equal decomposition of search key and index lookup
    - grouping according to resource-ID & counting
    - exact search: 10 matches (for Humboldt)
    - fuzzy search (fault tolerant, substrings): > 10 matches, however two-phase selection!
- **Nearest-neighbor search**
  - extension of the routing protocol
  - prerequisite: neighborhood-preserving hash function

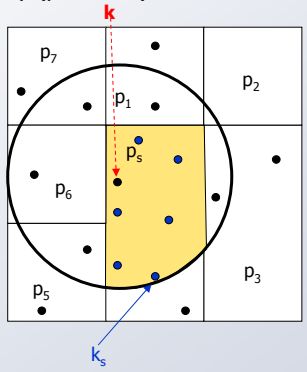
$$k = \left\{ \sum_{i=0}^{1/2} c_{2i,} \bullet v, \sum_{i=0}^{1/2} c_{2i+1} \bullet v \right\}$$

  - example: (v=0, ASCII codes):
    f("Beethoven") = {449, 429}
    f("Bethoven") = {394, 433}

Nearest-Neighbor Search (2)

- NN-query requesting the *s* nearest records for *k*
  - send query to p_s (responsible for *k)*
  - p_s delivers *a* number of records + list of neighbors
  - estimation of quality (precision)

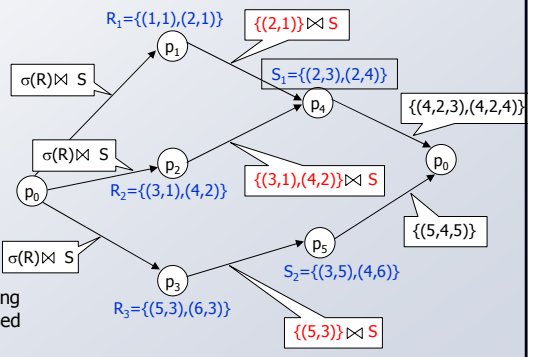
$$\frac{\text{volume of queried space}}{\text{volume of circle with radius dist}(k,k_s)}$$



  - result refinement directing query to neighbors of p_s

Evaluation Strategies

- **Goal: "stateless" evaluation**
  - several copies of the QEP "travel" through the network
    → no wait for input table
  - peer for following operation determined by hash value of the current intermediate results
- **Approaches**
  - pre-optimization
    - creation and stepwise processing of left-deep query trees
  - adaptive processing
    - selection of the following operator to be processed at runtime (Eddy mechanism)



An Eddy is a distributor for records. For each incoming record, a decision is made by which operator of the plan it is processed next.

Summary

- **Early projects**
- **Search in P2P systems**
  - routing indexes
- **Scalable distributed data structures**
  - Chord: data organization, search and dynamic operations
  - Content Addressable Network (CAN)
  - P-Grid
- **Distributed hash tables for structured data**
  - storage of relational data
  - query operators: lookup-, range- and join queries
  - similarity-based queries and robust query execution

Literature

- J. Ritter: Why Gnutella Can't Scale. No, Really. (http://www.tch.org/gnutella.html)
- A. Petersson: Gnutella. (http://www.petersson.at/gnutella)
- A. Crespo, H. Garcia-Molina: Routing Indices for Peer-to-Peer Systems, Proc. Int. Conf. on Distributed Computing, 2002
- I. Stoica, R. Morris, D. Karger, K. F. Kaashoek, H. Balakrishan: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, ACM SIGCOMM 2001
- S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker: A Scalable Content-Adressable Network, ACM SIGCOMM 2001
- K. Aberer: P-Grid: A Self-organizing Access Structure for P2P Information Systems, CoopIS 2001
- K. Aberer, M. Hauswirth: P2P Information Systems: Concepts and Models, State-of-the-art, and Future Systems (Tutorial), ICDE 2002 (http://www.p-grid.org)
- B. Kröll, P. Widmayer: Distributing a Search Tree Among a Growing Number of Processors, SIGMOD'94

Thanks to W. Lehner and K.-U. Sattler for the support!