SIMILARITY SEARCH
The Metric Space Approach

Pavel Zezula, Giuseppe Amato,
Vlastislav Dohnal, Michal Batko

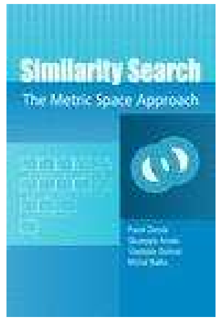


Table of Content

Part I: Metric searching in a nutshell

- Foundations of metric space searching
- Survey of existing approaches

Part II: **Metric searching in large collections**

- Centralized index structures
- Approximate similarity search
- **Parallel and distributed indexes**

Parallel and Distributed Indexes

1. **preliminaries**
2. processing M-trees with parallel resources
3. scalable and distributed similarity search
4. performance trials

Parallel Computing

- Parallel system
  - Multiple independent processing units
  - Multiple independent storage places
  - Shared dedicated communication media
  - Shared data
- Example
  - Processors (CPUs) share operating memory (RAM) and use a shared internal bus for communicating with the disks

Parallel Index Structures

- Exploiting parallel computing paradigm
- Speeding up the object retrieval
  - Parallel evaluations
    - using multiple processors at the same time
  - Parallel data access
    - several independent storage units
- Improving responses
  - CPU and I/O costs

Parallel Search Measures

- The degree of the parallel improvement
- **Speedup**
  - Elapsed time of a fixed job run on
    - a small system (ST)
    - a big system (BT)

$$speedup = \frac{ST}{BT}$$

  - Linear speedup
    - *n*-times bigger system yields a speedup of *n*

Parallel Search Measures

- **Scaleup**
  - Elapsed time of
    - a small problem run on a small system (STSP)
    - a big problem run on a big system (BTBP)

$$scaleup = \frac{STSP}{BTBP}$$

  - Linear scaleup
    - The *n*-times bigger problem on *n*-times bigger system is evaluated in the same time as needed by the original system to process the original problem size

Distributed Computing

- Parallel computing on several computers
  - Independent processing and storage units
    - CPUs and disks of all the participating computers
  - Connected by a network
    - High speed
    - Large scale
    - Internet, corporate LANs, etc.
- Practically unlimited resources

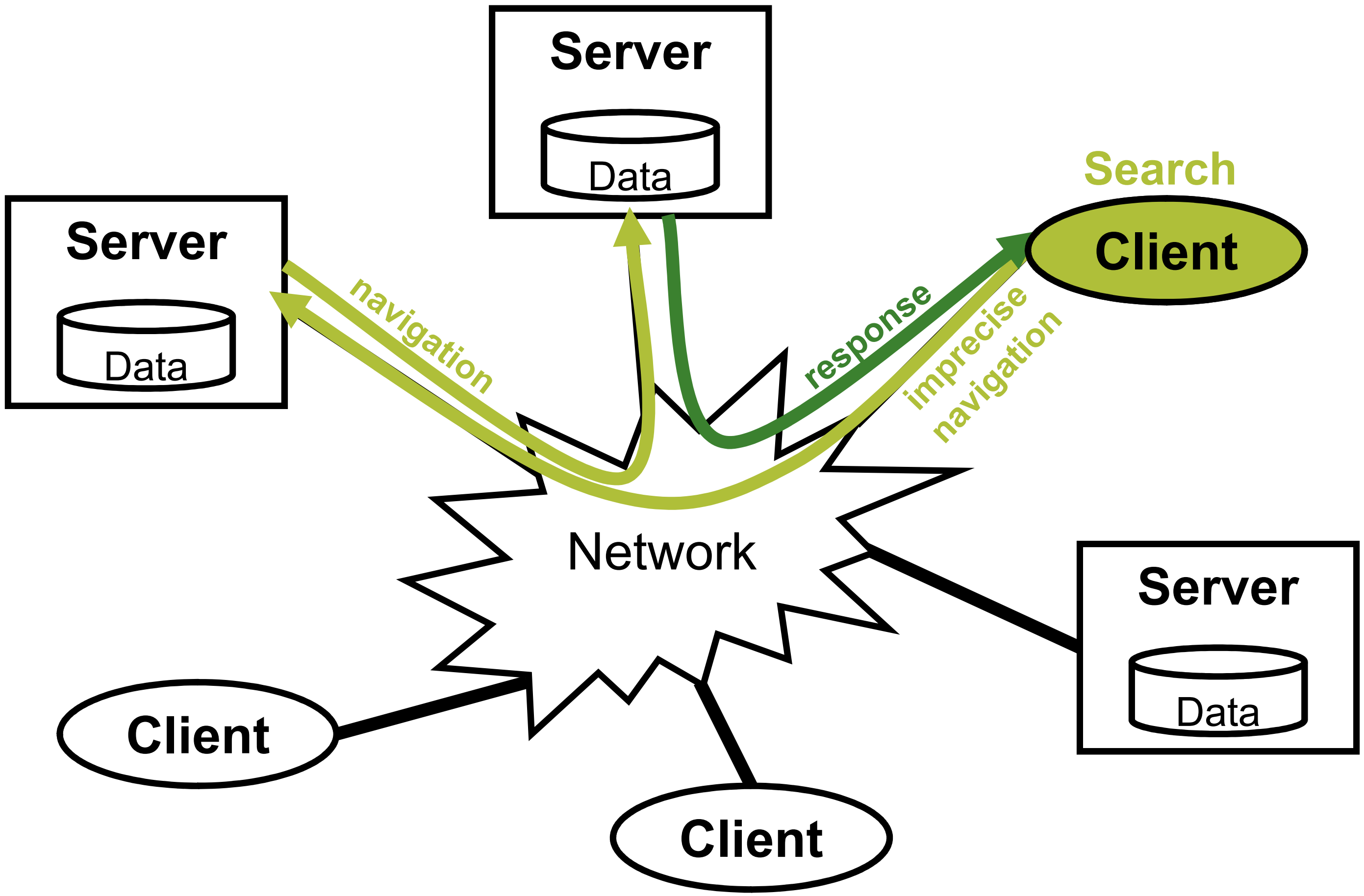
Distributed Index Structures

- Data stored on multiple computers
  - Navigation (routing) algorithms
- Solving queries and data updates
  - Network communication
- Efficiency and scalability
  - Scalable and Distributed Data Structures
  - Peer-to-peer networks

Scalable & Distributed Data Structures

- Client/server paradigm
  - Clients pose queries and update data
  - Servers solve queries and store data
- Navigation algorithms
  - Use local information
  - Can be imprecise
    - image adjustment technique to update local info

Distributed Index Example



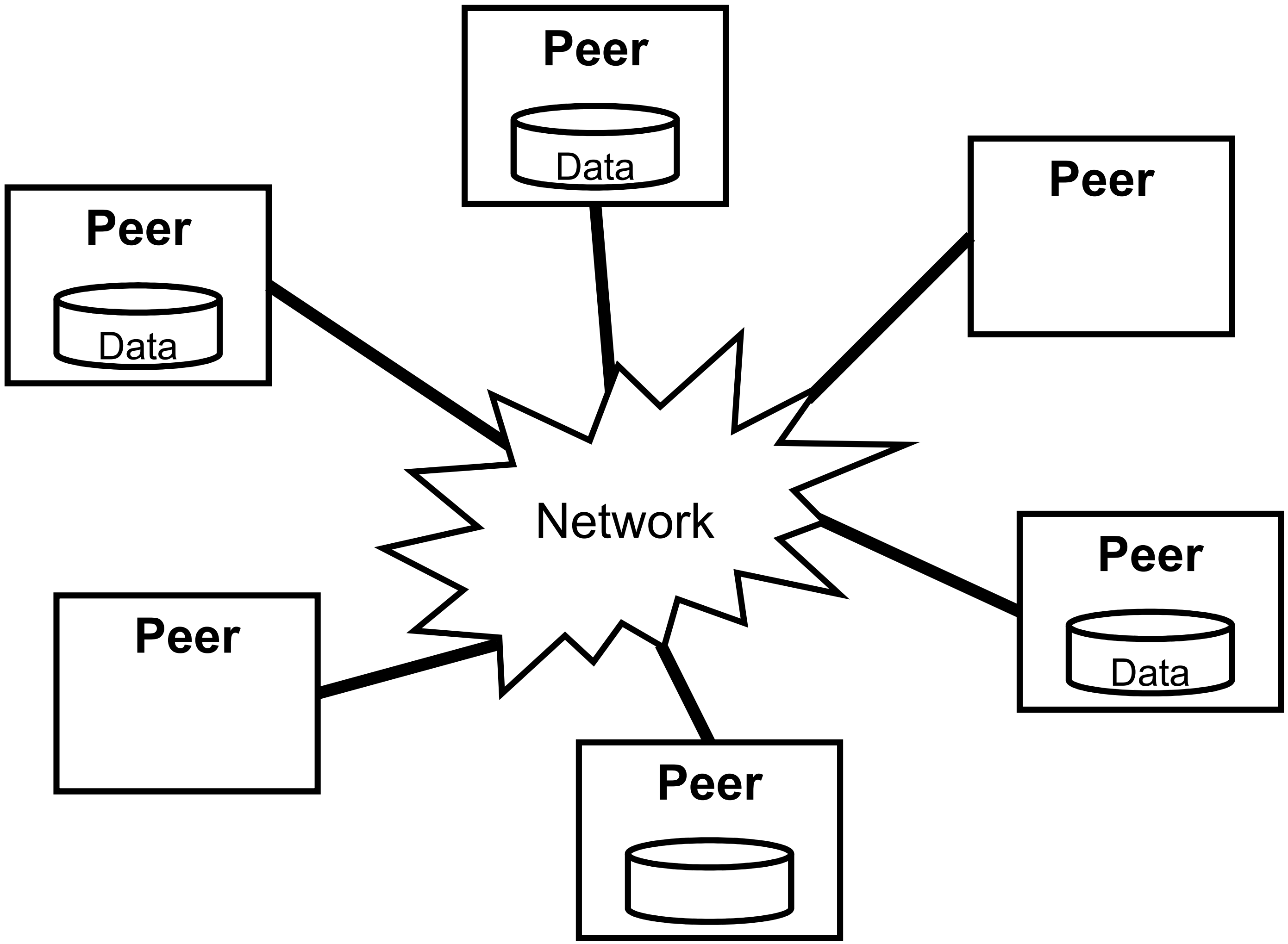
SDDS Properties

- **Scalability**
  - data migrate to new network nodes gracefully, and only when the network nodes already used are sufficiently loaded
- **No hotspot**
  - there is no master site that must be accessed for resolving addresses of searched objects, e.g., centralized directory
- **Independence**
  - the file access and maintenance primitives (search, insert, node split, etc.) never requires atomic updates on multiple nodes

Peer-to-Peer Data Networks

- Inherit basic principles of the SDDS
- Peers are equal in functionality
  - Computers participating in the P2P network have the functionality of both the client and the server
- Additional high-availability restrictions
  - Fault-tolerance
  - Redundancy

Peer-to-Peer Index Example



Parallel and Distributed Indexes

1. preliminaries
2. **processing M-trees with parallel resources**
3. scalable and distributed similarity search
4. performance trials

Processing M-trees with parallel resources

- Parallel extension to the basic M-Tree
  - To decrease both the I/O and CPU costs
  - Range queries
  - *k-NN* queries
- Restrictions
  - Hierarchical dependencies between tree nodes
  - Priority queue during the *k-NN* search

M-tree: Internal Node (reminder)

- Internal node consists of an **entry** for each subtree
- Each entry consists of:
  - Pivot: *p*
  - Covering *radius* of the sub-tree: r^c
  - Distance from *p* to *parent* pivot p^p: $d(p,p^p)$
  - Pointer to sub-tree: *ptr*

$$\boxed{\boxed{\langle p_1, r_1^c, d(p_1, p^p), ptr_1 \rangle} \boxed{\langle p_2, r_2^c, d(p_2, p^p), ptr_2 \rangle} \cdots \boxed{\langle p_m, r_m^c, d(p_m, p^p), ptr_m \rangle}}$$

  - All objects in the sub-tree *ptr* are within the distance r^c from *p.*

M-tree: Leaf Node (reminder)

- Leaf node contains **data entries**
- Each entry consists of pairs:
  - Object (its identifier): *o*
  - Distance between *o* and its parent pivot: $d(o,o^p)$

$$\boxed{\boxed{\langle o_1, d(o_1,o^p)\rangle}\boxed{\langle o_2, d(o_2,o^p)\rangle}\cdots\boxed{\langle o_m, d(o_m,o^p)\rangle}}$$

Parallel M-Tree: Lowering CPU costs

- Inner node parallel acceleration
  - Node on given level cannot be accessed
    - Until all its ancestors have been processed
  - Up to *m* processors compute distances to pivots $d(q,p_i)$

$$\boxed{\boxed{\langle p_1, r_1^c, d(p_1,p^p), ptr_1\rangle}\boxed{\langle p_2, r_2^c, d(p_2,p^p), ptr_2\rangle}\cdots\boxed{\langle p_m, r_m^c, d(p_m,p^p), ptr_m\rangle}}$$

- Leaf node parallel acceleration
  - Independent distance evaluation $d(q,o_i)$ for all leaf objects

$$\boxed{\boxed{\langle o_1, d(o_1,o^p)\rangle}\boxed{\langle o_2, d(o_2,o^p)\rangle}\cdots\boxed{\langle o_m, d(o_m,o^p)\rangle}}$$

- *k-NN* query priority queue
  - One dedicated CPU

Parallel M-Tree: Lowering I/O costs

- Node accessed in specific order
  - Determined by a specific similarity query
  - Fetching nodes into main memory (I/O)
- Parallel I/O for multiple disks
  - Distributing nodes among disks
  - **Declustering** to maximize parallel fetch
    - Choose disk where to place a new node (originating from a split)
    - Disk with as few nodes with similar objects/regions as possible is a good candidate.

Parallel M-Tree: Declustering

- Global allocation declustering
  - Only number of nodes stored on a disk taken into account
    - Round robin strategy to store a new node
    - Random strategy
  - No data skew
- Proximity-based allocation declustering
  - Proximity of nodes' regions determine allocation
  - Choose the disk with the lowest sum of proximities
    - between the new node region
    - and all the nodes already stored on the disk

Parallel M-Tree: Efficiency

- Experimental evaluation
  - Good speedup and scaleup
  - Sequential components not very restrictive
- Linear speedup on CPU costs
  - Adding processors linearly decreased costs
- Nearly constant scaleup
  - Response time practically the same with
    - a five times bigger dataset
    - a five times more processors
  - Limited by the number of processors

Parallel M-Tree: Object Declustering

- Declusters objects instead of nodes
  - Inner M-Tree nodes remain the same
  - Leaf nodes contain pointers to objects
    - Objects get spread among different disks
- Similar objects are stored on different disks
  - Objects accessed by a similarity query are maximally distributed among disks
    - Maximum I/O parallelization
  - Range query $R(o_N,d(o_N,p))$ while inserting o_N
    - Choose the disk for physical storage
      - with the minimum number of retrieved objects

Parallel and Distributed Indexes

1. preliminaries
2. processing M-trees with parallel resources
3. **scalable and distributed similarity search**
4. performance trials

Distributed Similarity Search

- Metric space indexing technique
  - Generalized hyper-plane partitioning
- Peer-to-Peer paradigm
  - Self organizing
  - Fully scalable
  - No centralized components

GHT* Structure

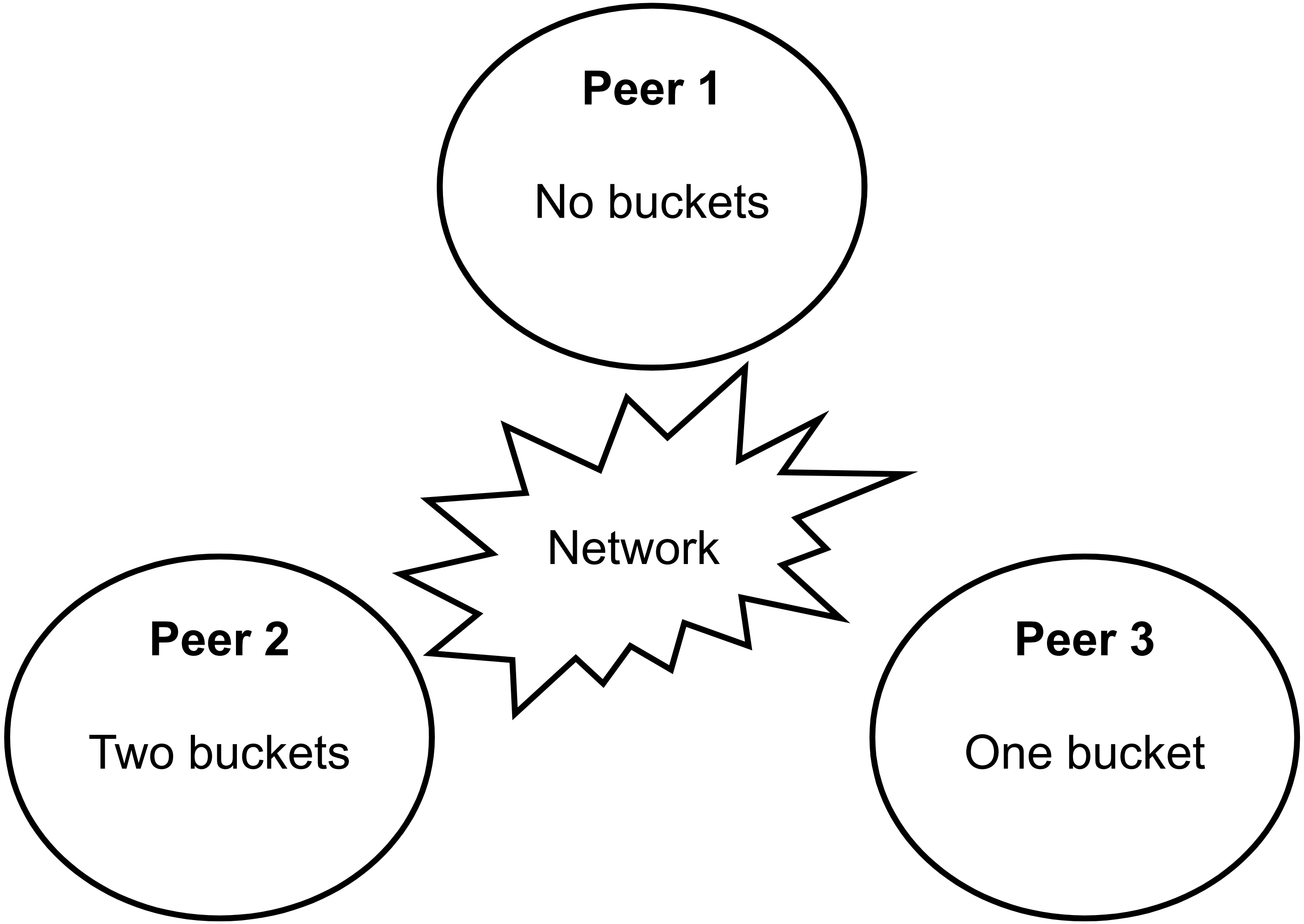
GHT* Architecture

- Peers
  - Computers connected by the network
    - message passing paradigm
    - request and acknowledgment messages
  - Unique (network node) identifier *NNID*
  - Issue queries
  - Insert/update data
  - Process data and answer queries

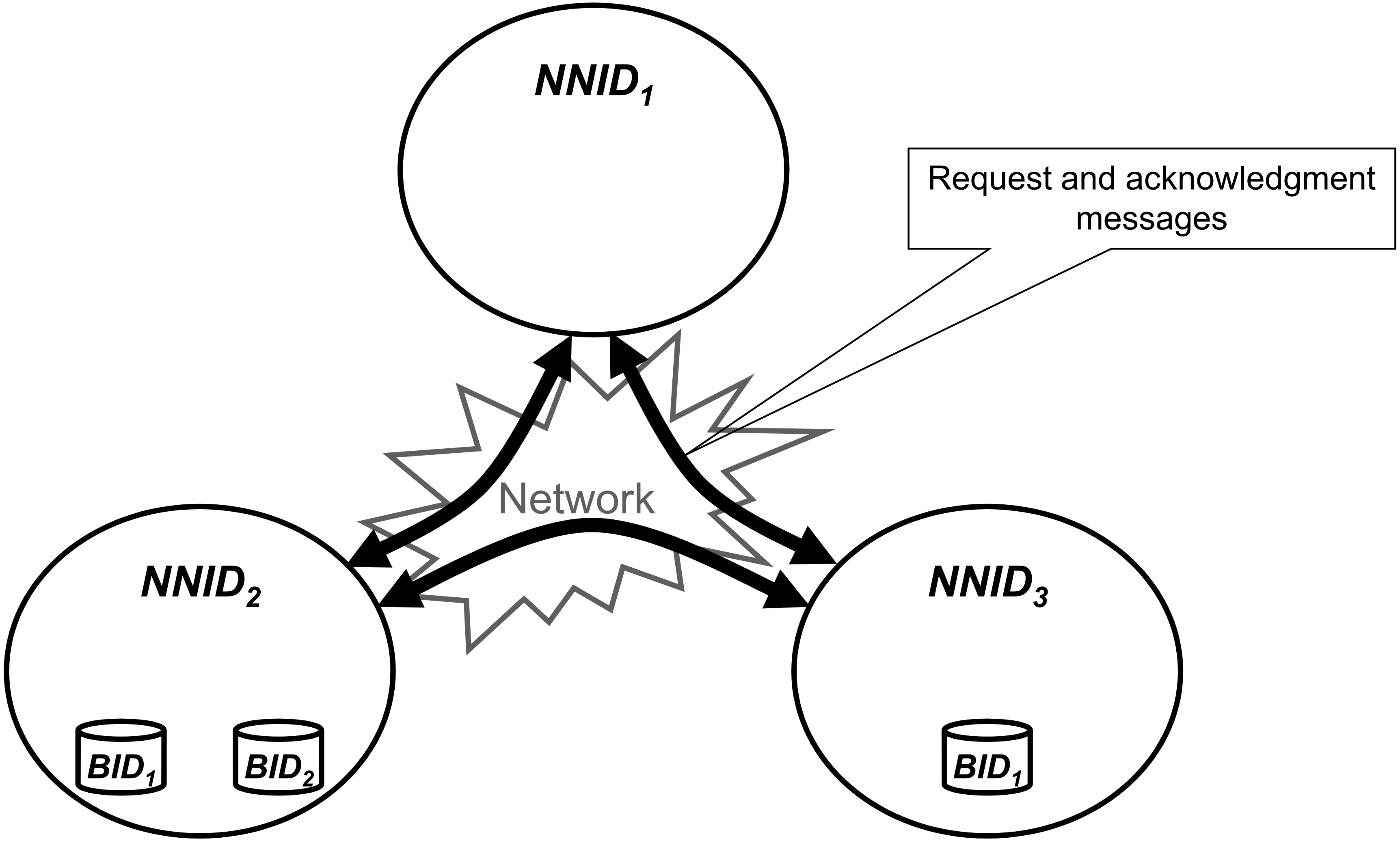
GHT* Architecture (cont.)

- Buckets
  - Storage for data
    - metric space objects
    - no knowledge about internal structure
  - Limited space
    - Splits/merges possible
  - Held by peers, multiple buckets per peer
    - there can be no bucket in a peer
    - identified by *BID,* unique within a peer

GHT* Architecture Schema



GHT* Architecture Schema (cont.)

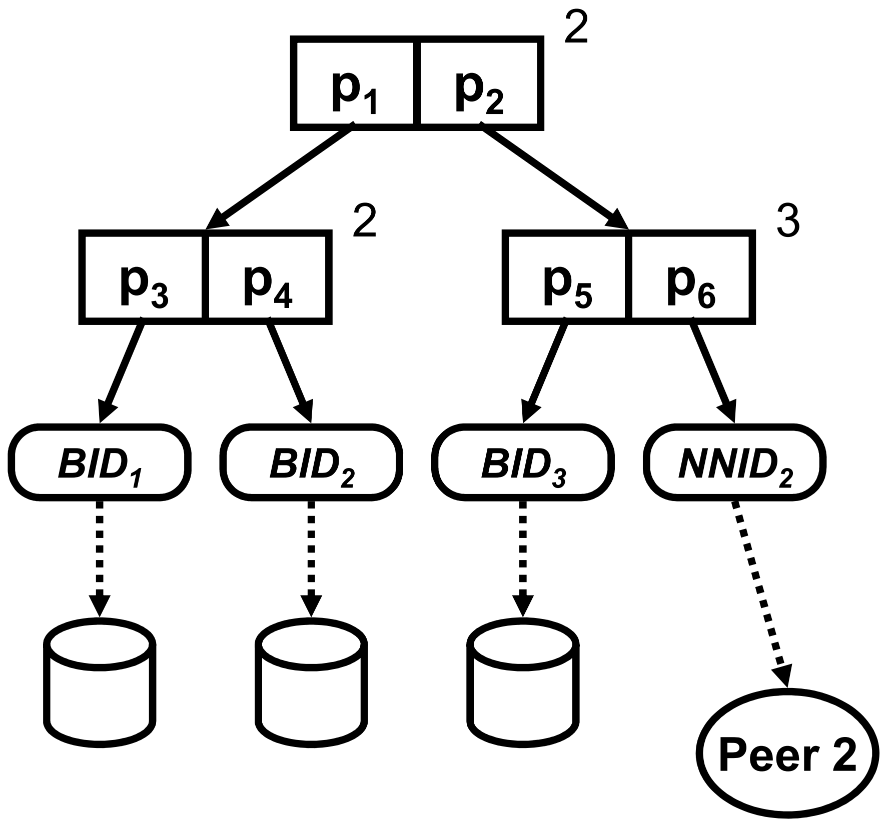


GHT* Architecture (cont.)

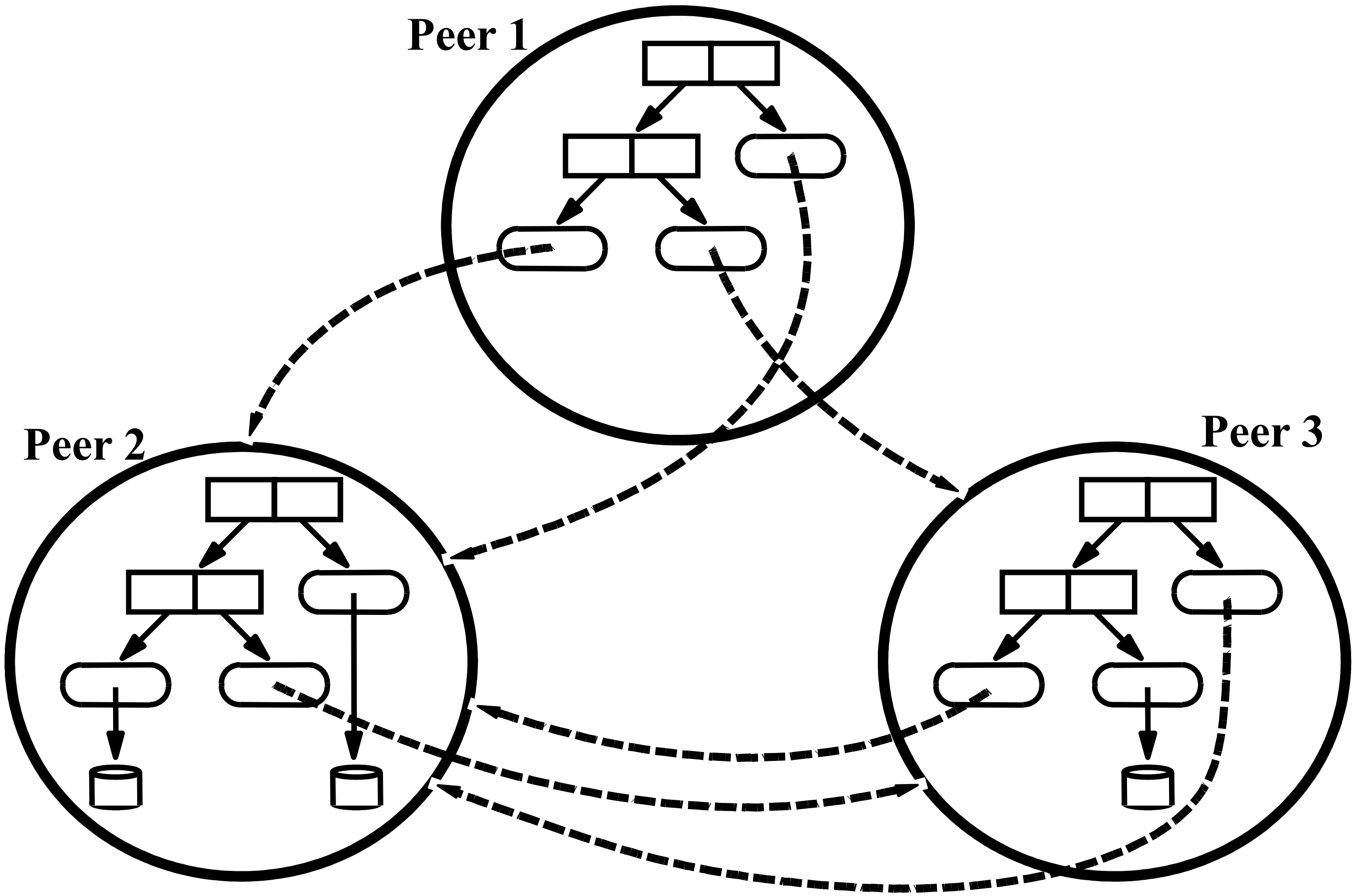
- Precise location of every object
  - Impossible to maintain on every peer
  - Navigation needed in the network
- Address search tree (AST)
  - Present in every peer
  - May be imprecise
    - repeating navigation in several steps
    - image adjustment

GHT* Address Search Tree

- Based on Generalized Hyperplane Tree
- Binary tree
- Inner nodes
  - pairs of pivots
  - serial numbers
- Leaf nodes
  - *BID* pointers to buckets
  - *NNID* pointers to peers

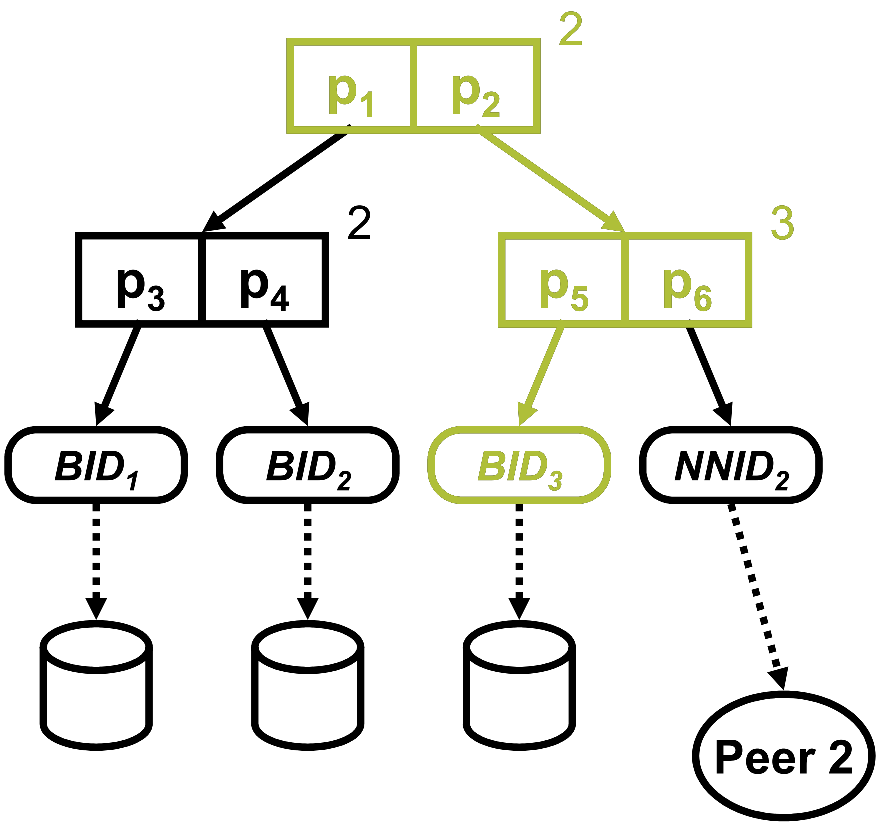


GHT* Address Search Tree



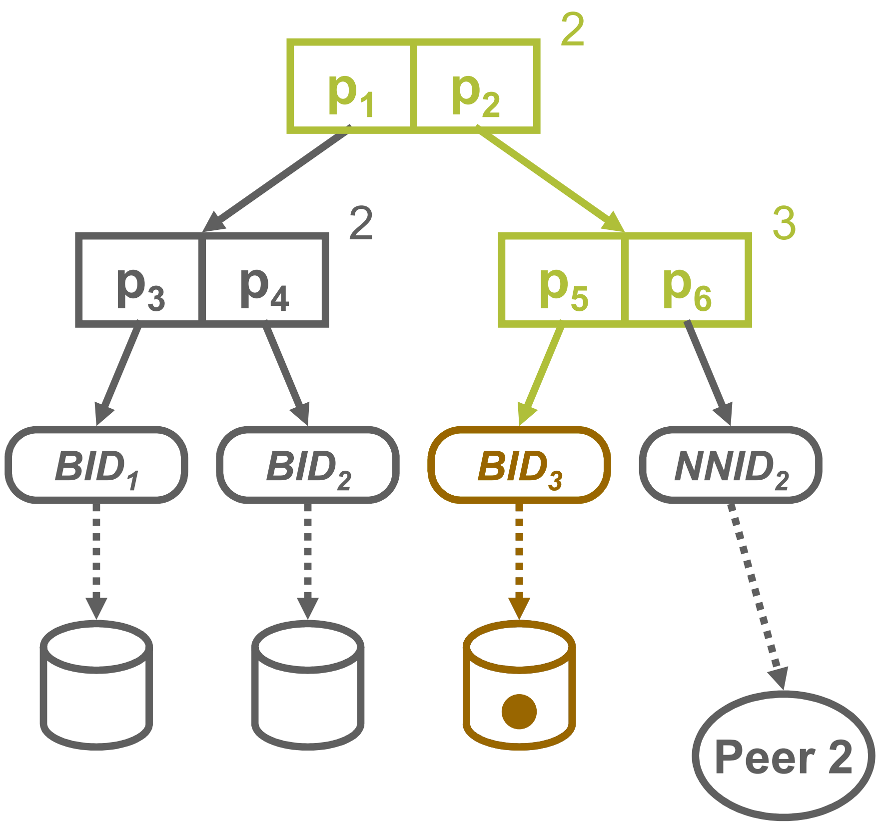
GHT* Inserting Objects

- ***Peer 1*** starts inserting an object ***o***
  - Use local AST
  - Start from the root
  - In every inner node:
    - take right branch if
      $d(p_1,o) > d(p_2,o)$
    - take left branch if
      $d(p_5,o) \le d(p_6,o)$
  - Till a leaf node is reached



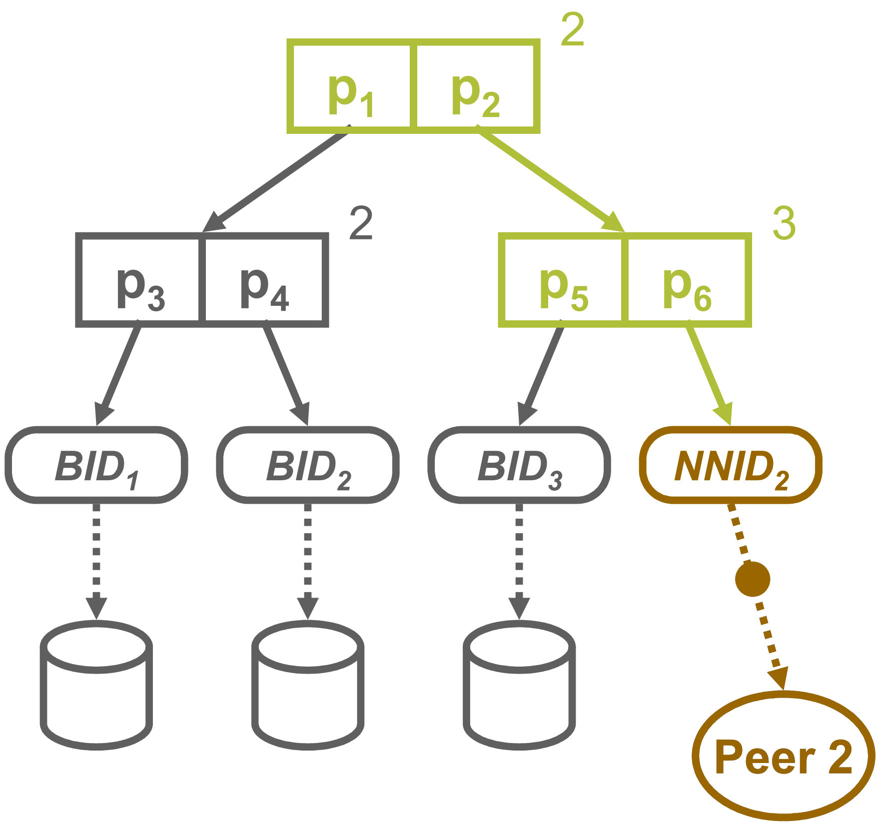
GHT* Inserting Objects (cont.)

- ***Peer 1*** inserting the object ***o***
  - If a *BID* pointer is found
    - Store the object ***o*** into the pointed bucket
    - The bucket is local (stored on ***peer 1***)



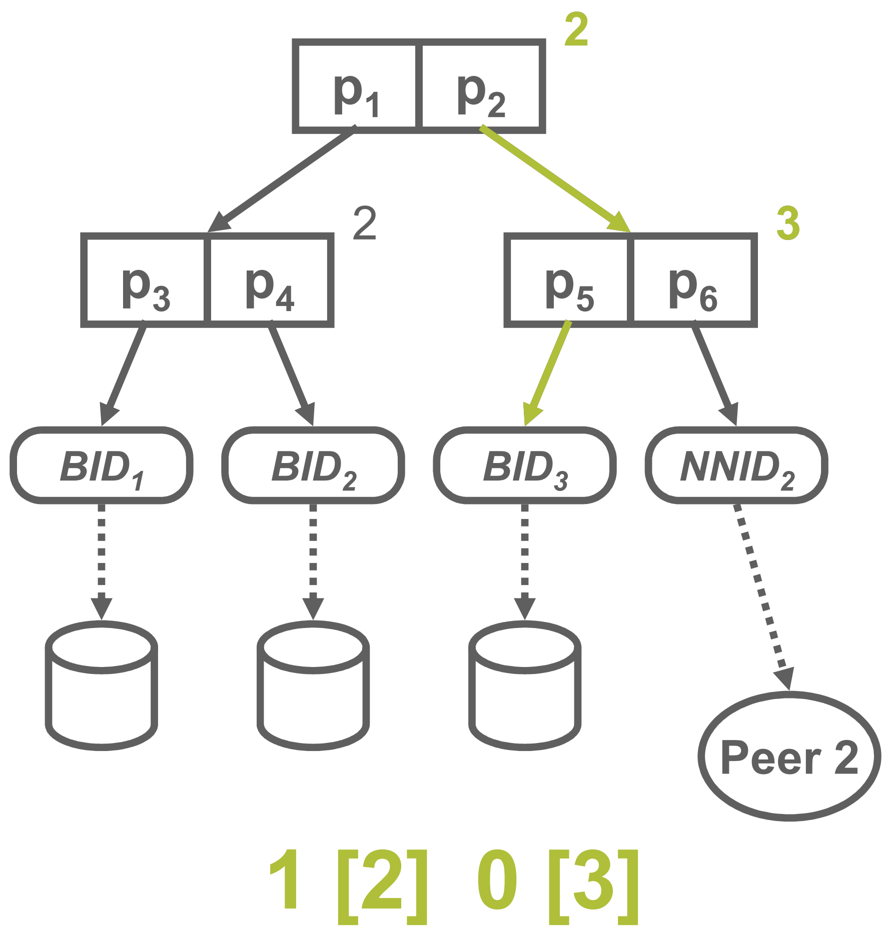
GHT* Inserting Objects (cont.)

- ***Peer 1*** inserting the object ***o***
  - If an *NNID* pointer is found
    - The inserted object ***o*** is sent to ***peer 2***
    - Where the insertion resumes



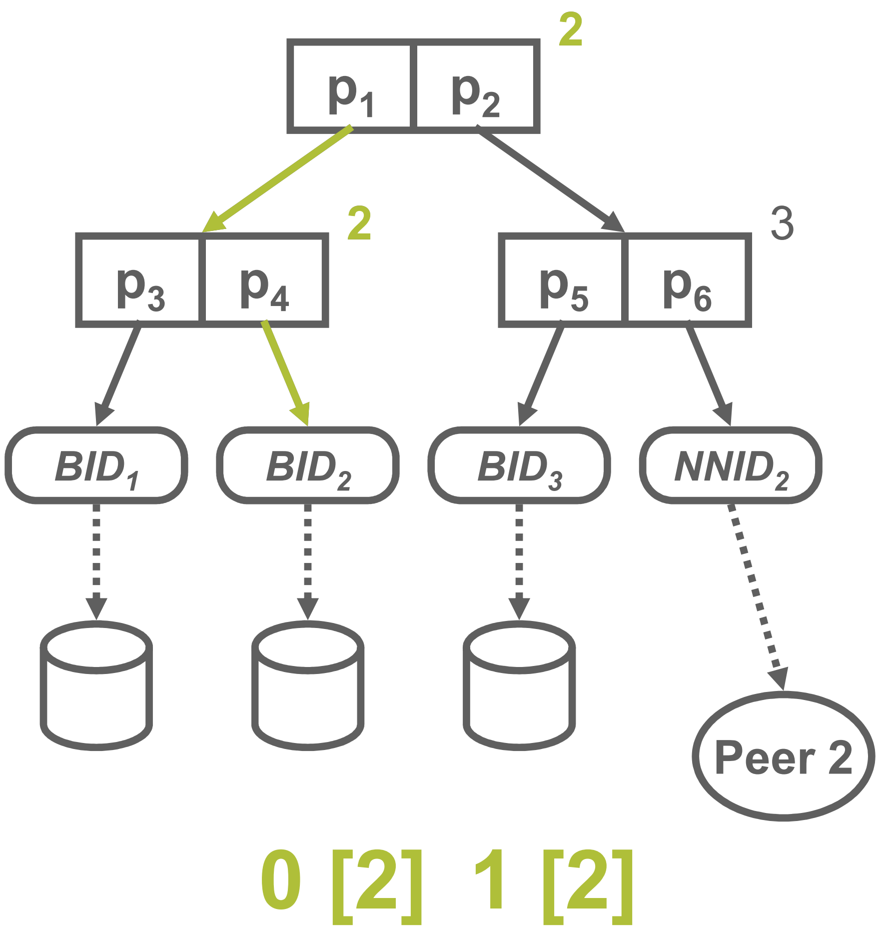
GHT* Binary Path

- Represents an AST traversal path
- String of *ones* and *zeros*
  - '0' means left branch
  - '1' means right branch
- Serial numbers
  - in inner nodes
  - detect obsolete parts
- Traversal example:



GHT* Binary Path (cont.)

- Example of a different path



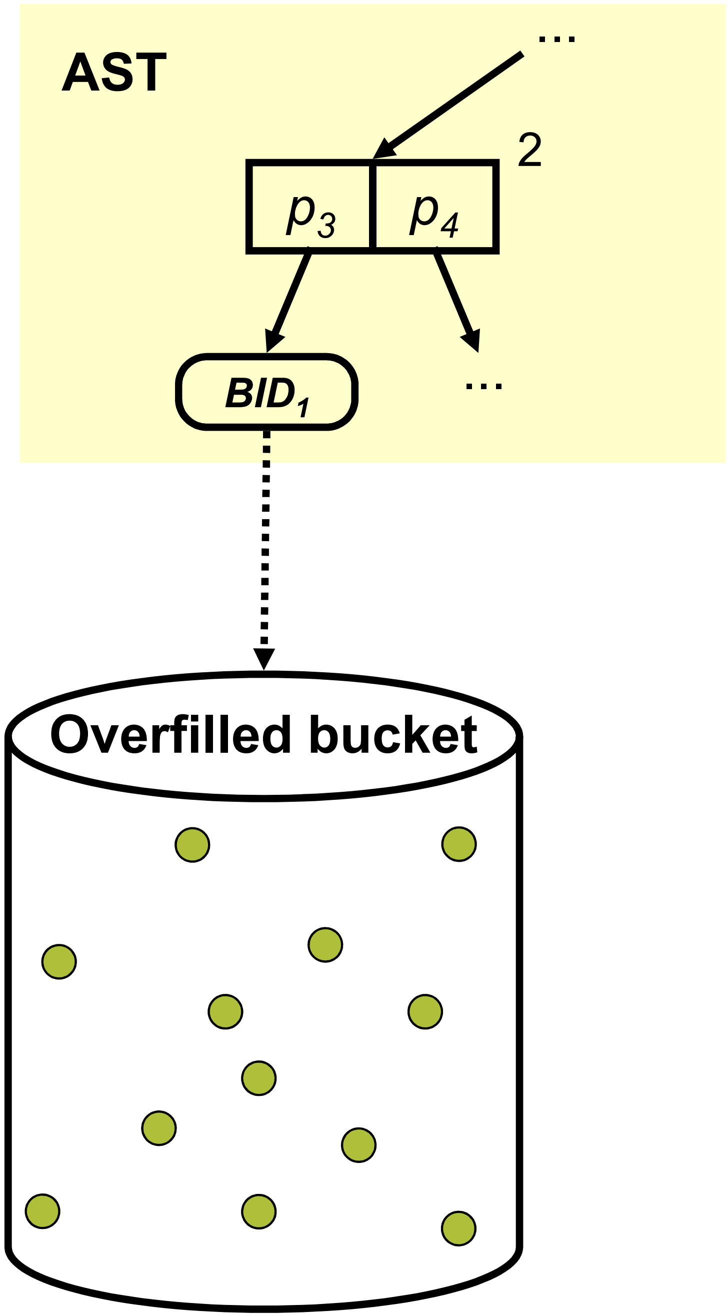
GHT* Storage Management

- Database grows as new data are inserted
- Buckets have limited capacity

- **Bucket splits**
  - Allocate a new bucket
  - Extend routing information
    - choose new pivots
  - Move objects

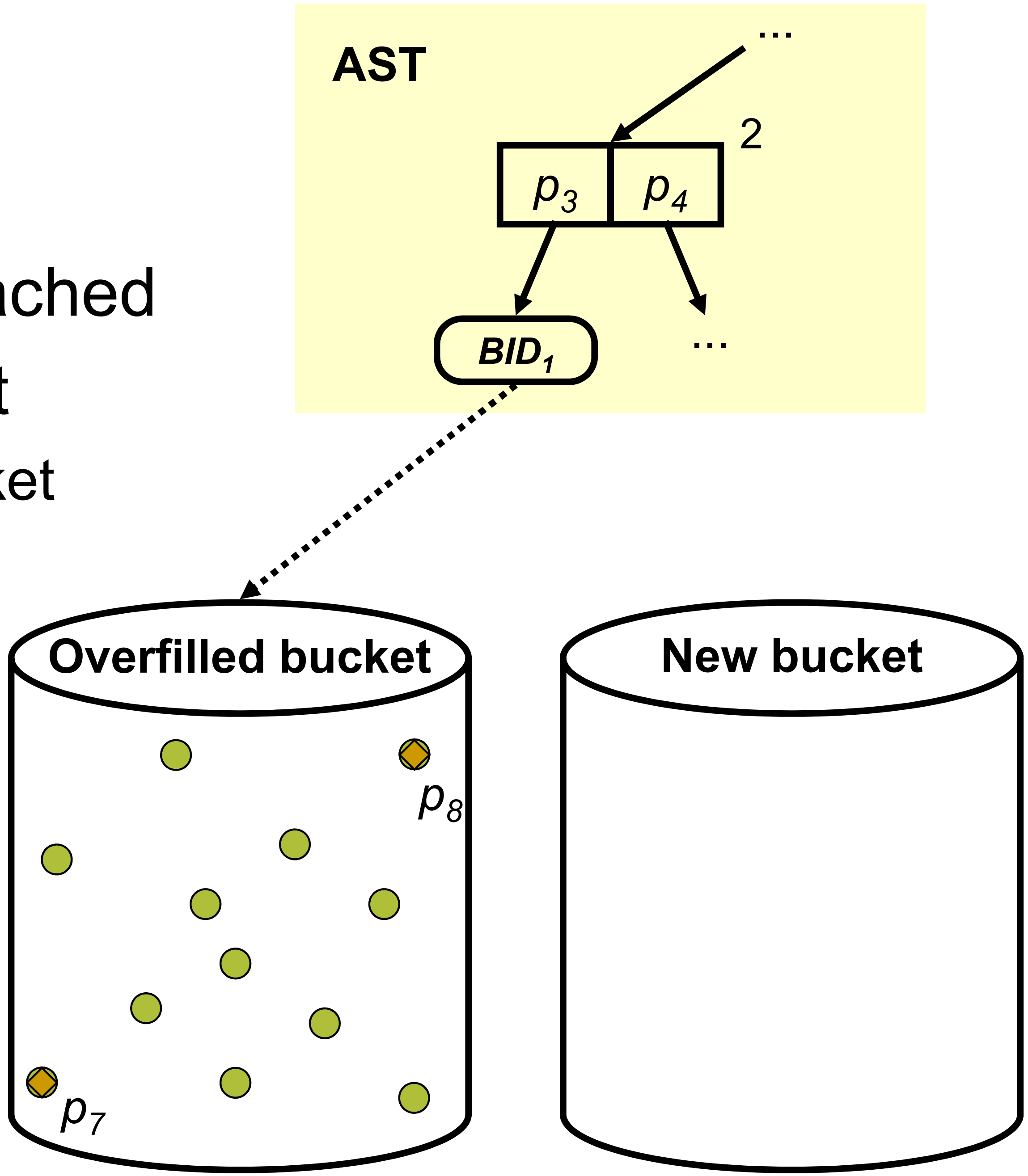
Splitting

- Bucket capacity is reached
- Allocate a new bucket
  - Either a new local bucket
  - or at another peer



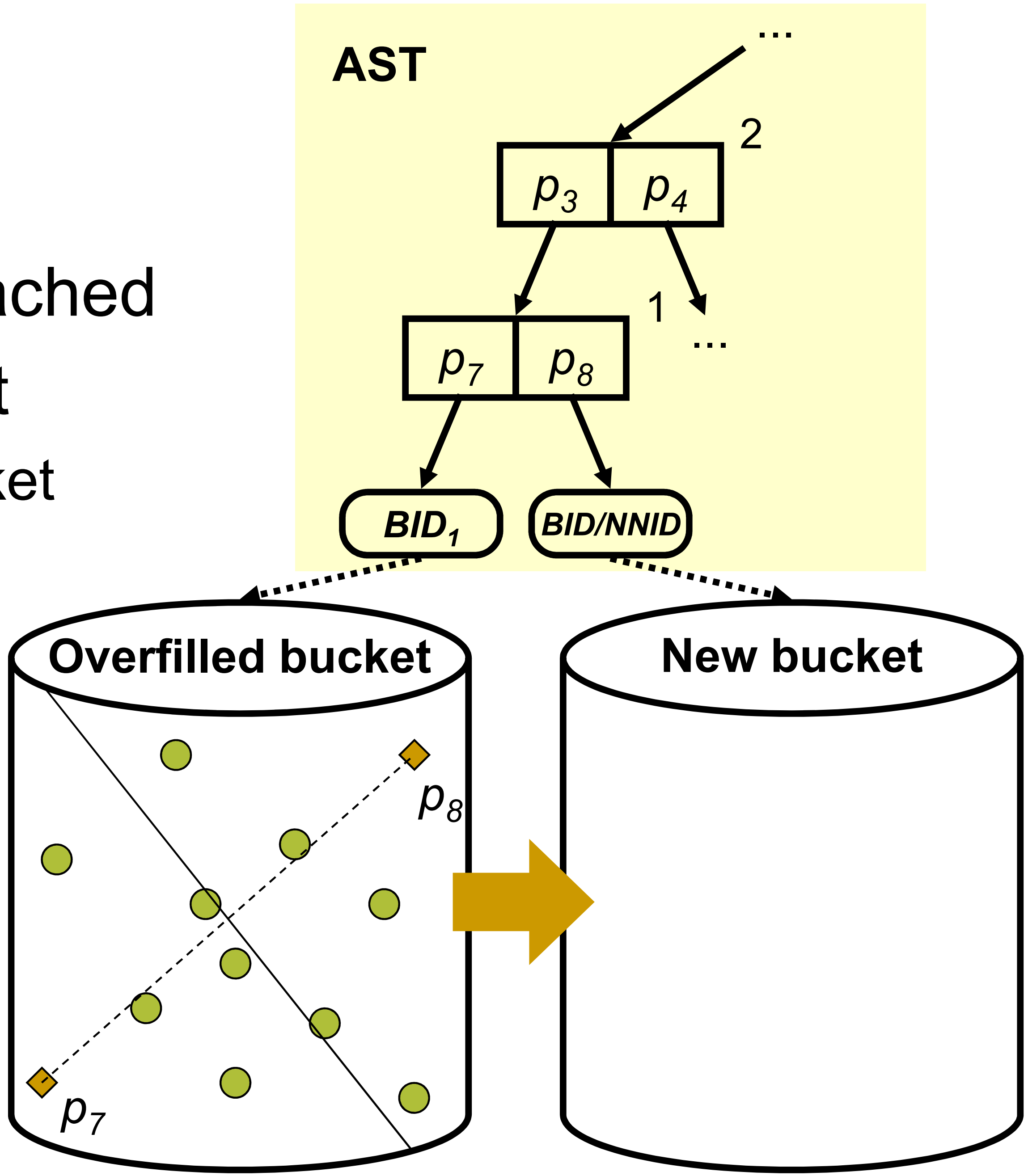
Splitting

- Bucket capacity is reached
- Allocate a new bucket
  - Either a new local bucket
  - or at another peer
- Choose new pivots
- Adjust AST



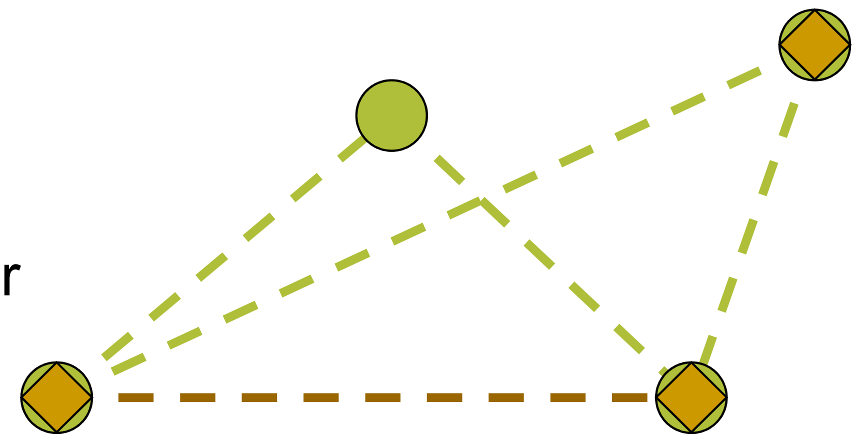
Splitting

- Bucket capacity is reached
- Allocate a new bucket
  - Either a new local bucket
  - or at another peer
- Choose new pivots
- Adjust AST
  - Inner node with pivots
  - Leaf node for the new bucket
- Move objects



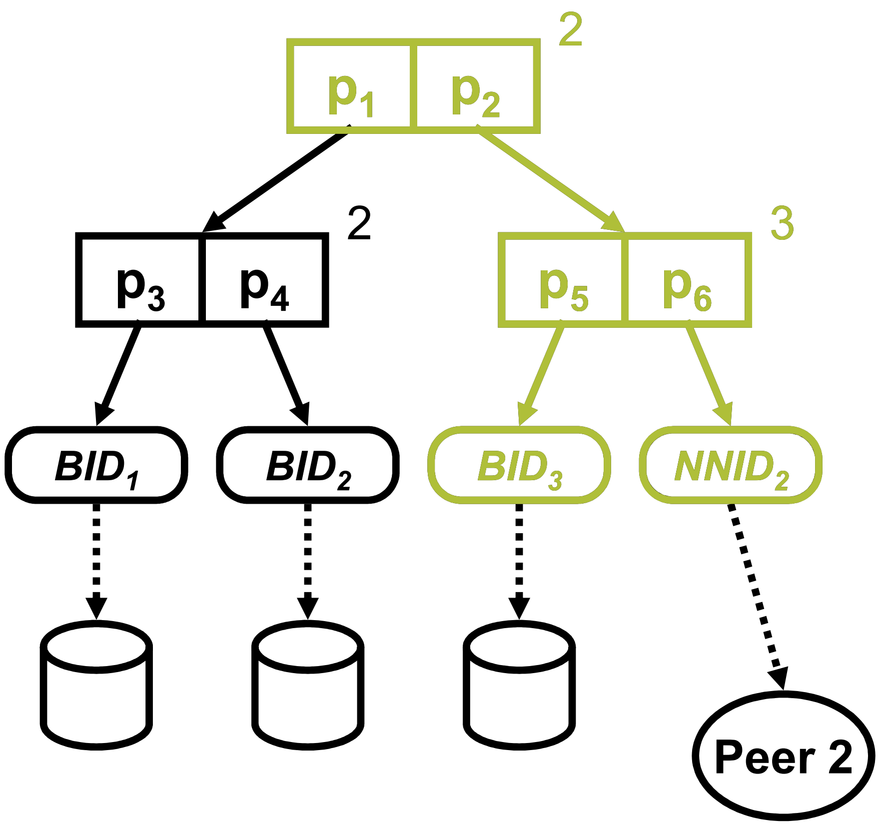
Pivot Choosing Algorithm

- Pivots are pre-selected during insertion
  - Two objects are marked at any time
  - The marked objects become pivots on split
- Heuristic to maximize the distance between pivots
  - Mark the first two inserted objects
  - Whenever a new object arrives
    - Compute its distances from the currently marked objects
    - If one of the distances is greater than the distance between marked objects
      - change the marked objects



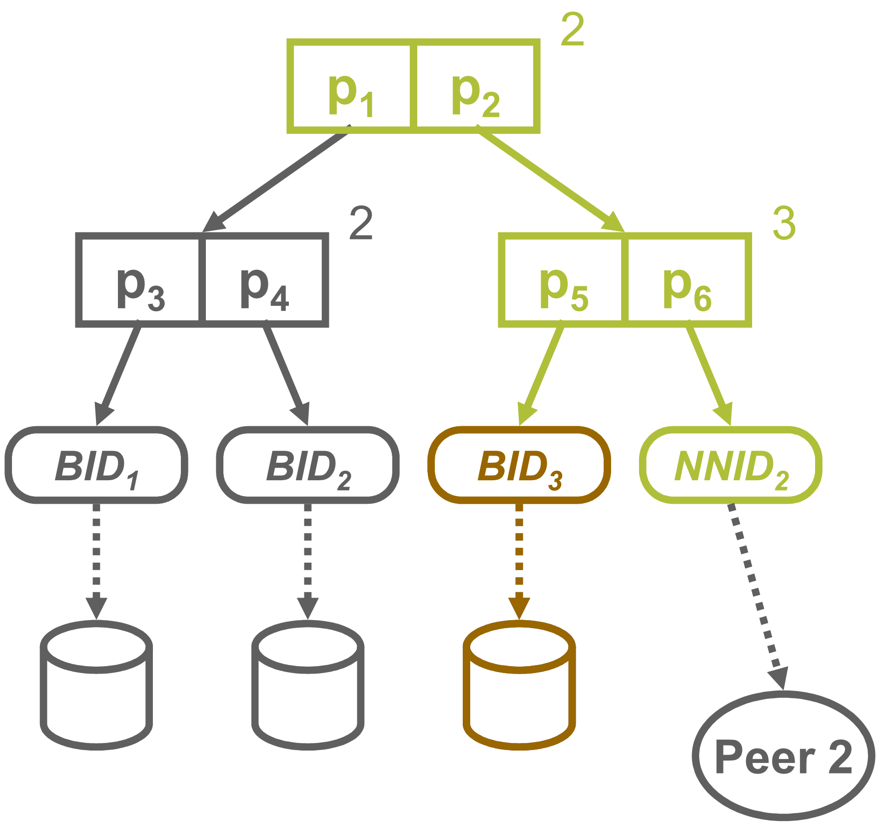
GHT* Range Search

- ***Peer 1*** starts evaluating a query ***R(q,r)***
  - Use the local AST
  - Start from the root
  - In each inner node:
    - take right branch if
      $d(p_a,q)+r > d(p_b,q)-r$
    - take left branch if
      $d(p_a,q)-r \le d(p_b,q)+r$
    - both branches can qualify
  - Till a leaf node is reached in each followed path



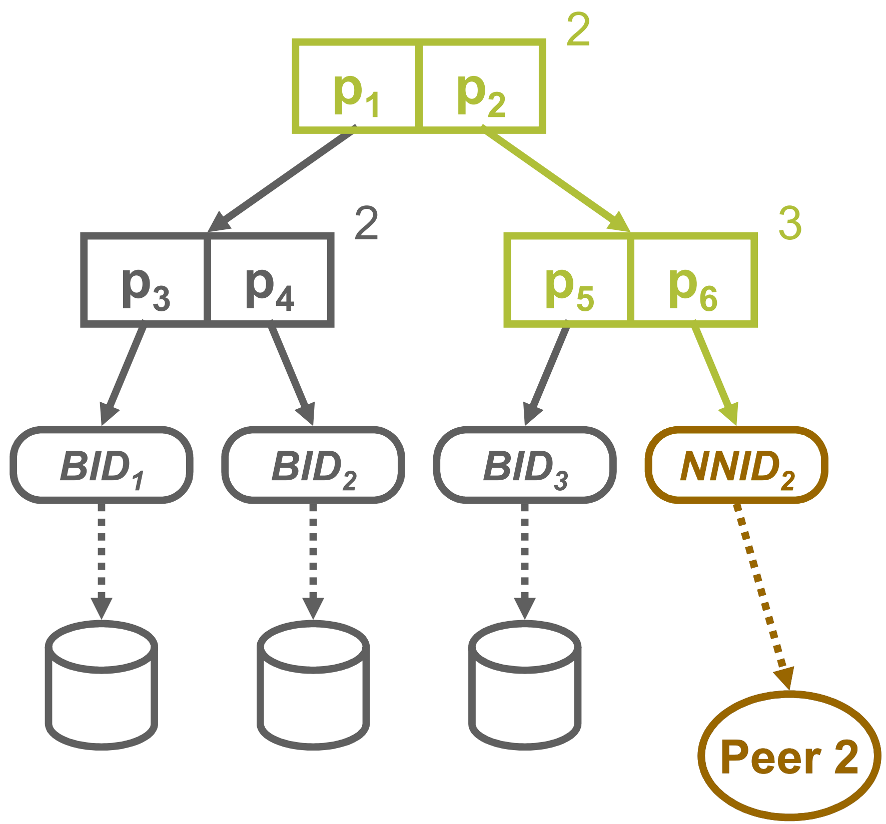
GHT* Range Search (cont.)

- ***Peer 1*** evaluating the range query ***R*(*q*,*r*)**
  - For every *BID* pointer found
    - Search the corresponding local bucket
    - Retrieve all objects *o* in the bucket that satisfy
      $$d(q,o) \leq r$$
    - Any centralized similarity search method can be used



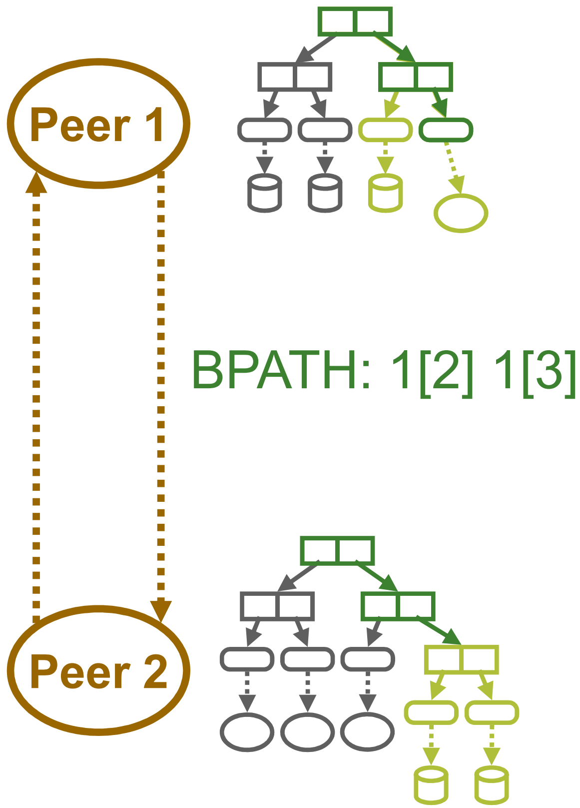
GHT* Range Search (cont.)

- ***Peer 1*** evaluating the range query ***R*(*q*,*r*)**
  - For every *NNID* pointer found
    - Continue with the search at corresponding peers



GHT* Range Search (cont.)

- ***Peer 1*** evaluating the range query ***R(q,r)***
  - For every *NNID* pointer found
    - Continue with the search at corresponding peers
      - Build *BPATH* for the traversal
      - Forward the message
    - Destination peers consult their ASTs
      - Avoid repeated computations using the *BPATH*
    - Wait until the results are gathered from all active peers
    - Merge them with results from local buckets



GHT* Nearest Neighbor Search

- Based on the range search
  - Estimate the query radius
- Evaluate *k*-nearest neighbors query ***k-NN*(*q*)**
  - Locate a bucket where ***q*** would be inserted
    - use the strategy for inserting an object
  - Start a range query with radius ***r*** equal to the distance between ***q*** and the ***k***-th nearest neighbor of ***q*** in this bucket
    - If the bucket contains less than ***k*** objects, estimate *r* using:
      - an optimistic strategy
      - an pessimistic strategy
  - The result of the range query contains the *k-NN* result

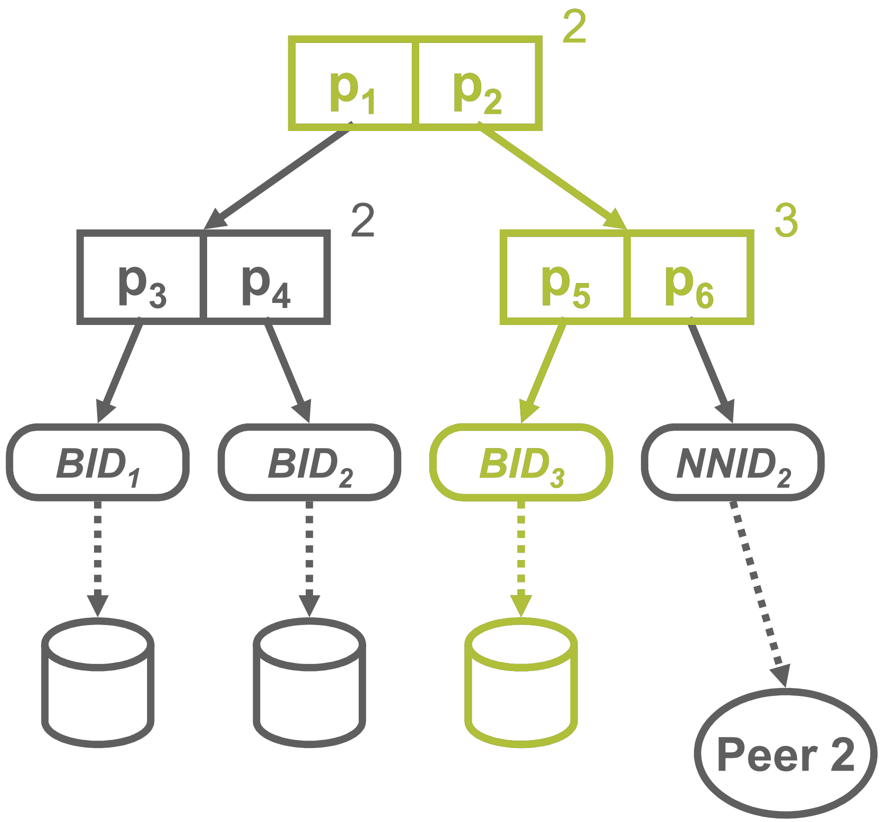
GHT* k-NN Search Example

- Example ***5-NN(q)***
  - Use the insert strategy in the local AST

$$d(p_1, q) > d(p_2, q)$$

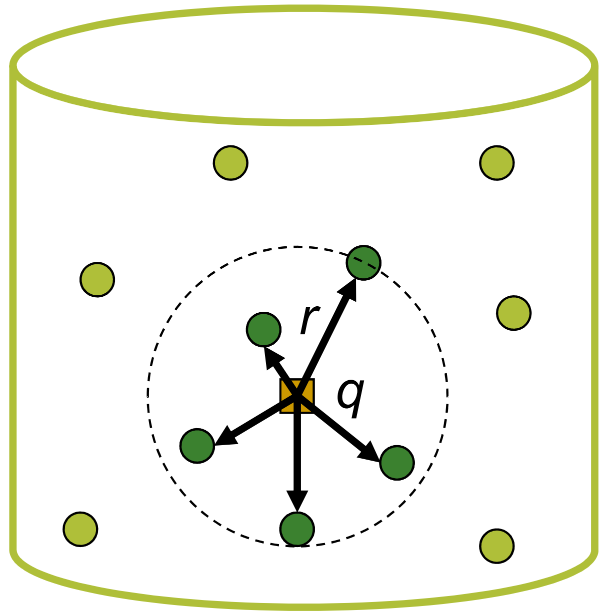
$$d(p_5, q) \le d(p_6, q)$$

  - Until a BID pointer is found
    - Continue searching at other peer whenever an NNID pointer is found
  - Search the destination bucket



GHT* *k*-NN Search Example (cont.)

- Example ***5-NN*(*q*)**
  - Retrieve five nearest neighbors of ***q*** in the local bucket
  - Set *r* to the distance of the fifth nearest neighbor found
  - Evaluate a distributed range search ***R*(*q*,*r*)**
    - results include at least five nearest neighbors from the local bucket
    - however, some additional objects closer to ***q*** can be found
  - Get the first five nearest objects of ***R*(*q*,*r*)**

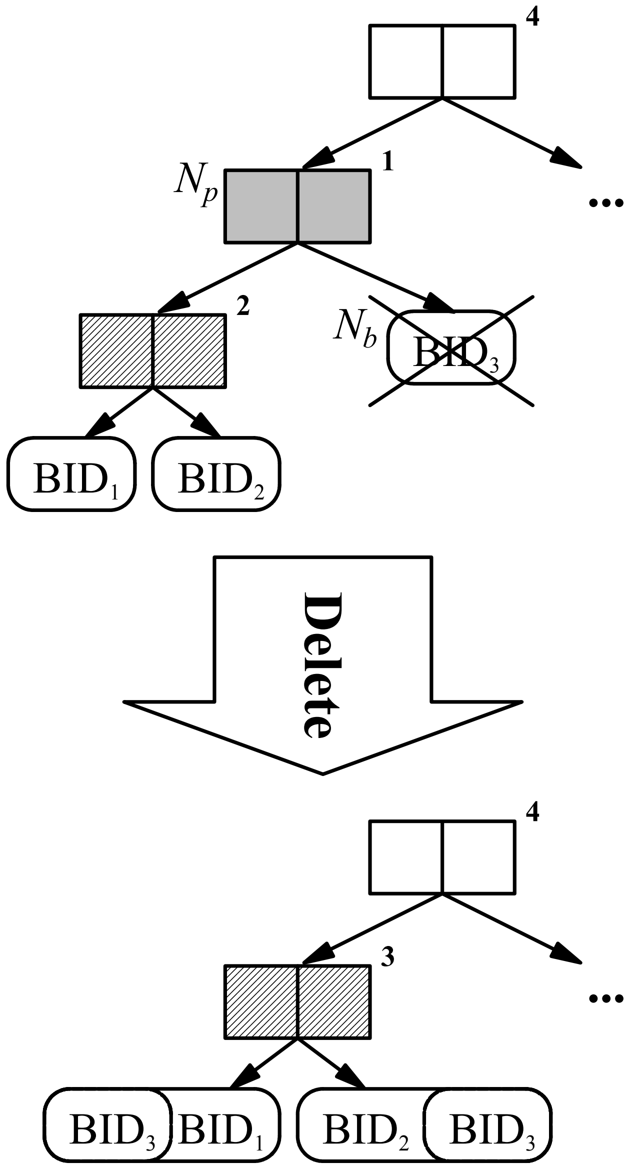


GHT* Updates and Deletions

- Updating an object
  - Delete the original object
  - Insert the updated version
- Deleting an object
  - Locate the bucket where the object is stored
    - the insert navigation algorithm is used
  - Remove the object from the bucket
  - The bucket occupation may become too low
    - merge the bucket with another one
    - update the corresponding nodes in the AST

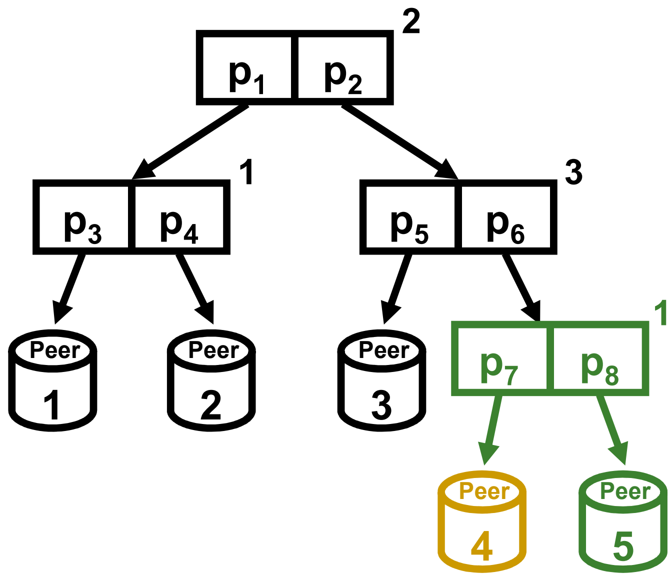
GHT* Merging Buckets

- Remove a bucket
  - Get its sibling
    - either a leaf node (bucket)
    - or an inner node
  - Reinsert all remaining objects
    - into the sibling
      - *multiple buckets possibly*
- Remove the inner node N_p
- Increase the node's serial number



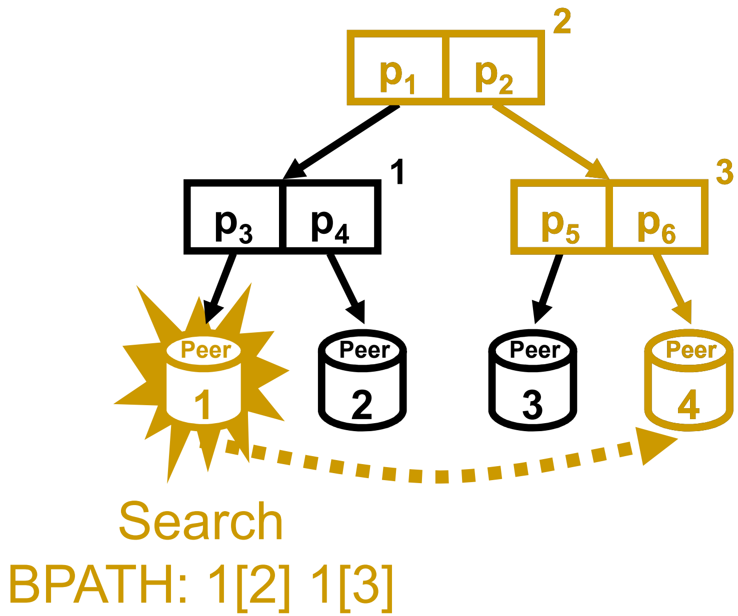
AST: Image Adjustment

- The AST is modified on bucket splits and merges
  - Only changed peers are aware of the change (4 and 5)



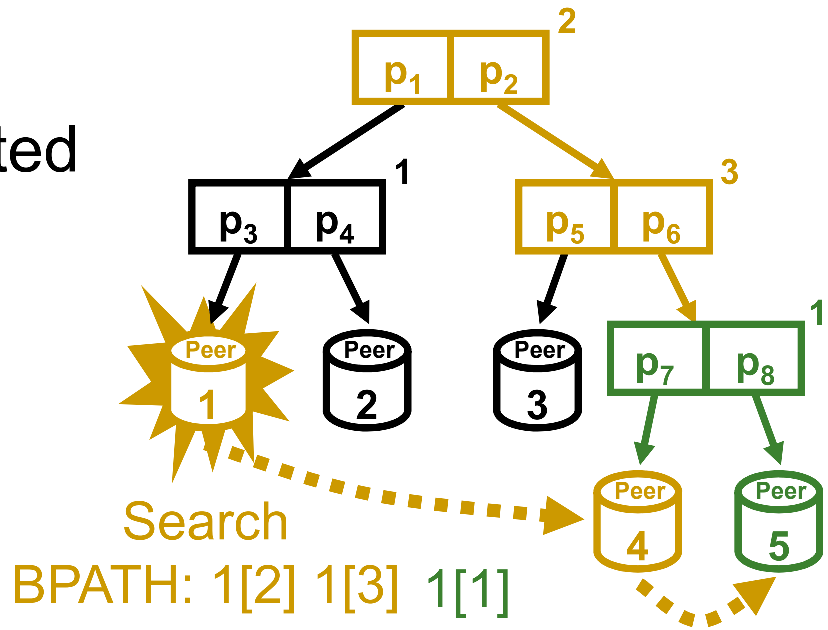
AST: Image Adjustment (cont.)

- The AST is modified on bucket splits and merges
  - Only changed peers are aware of the change (4 and 5)
- When other peer searches
  - Forwards the query to a peer



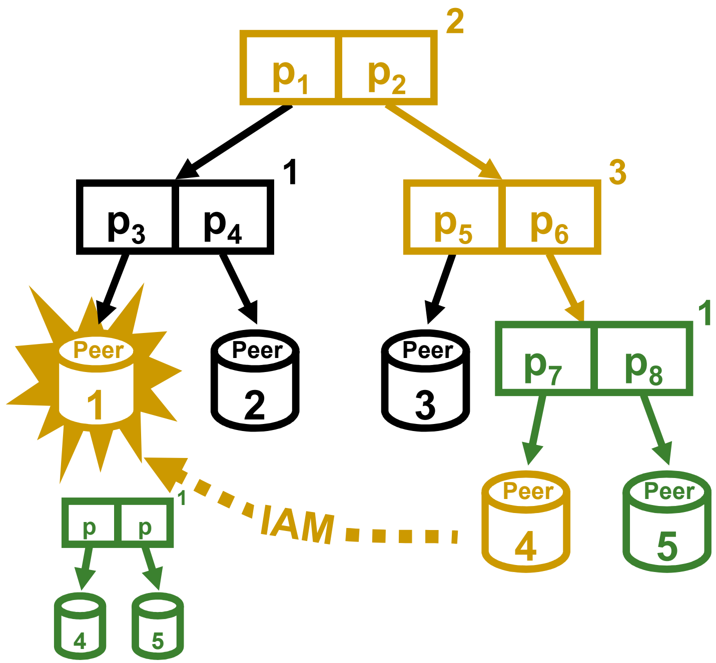
AST: Image Adjustment (cont.)

- The AST is modified on bucket splits and merges
  - Only changed peers are aware of the change (4 and 5)
- When other peer searches
  - Forwards the query to a peer
    - which has a different AST view
  - The incomplete search is detected
    - by too short *BPATH*
  - The search evaluation resumes
    - possibly forwarding the query to some other peers



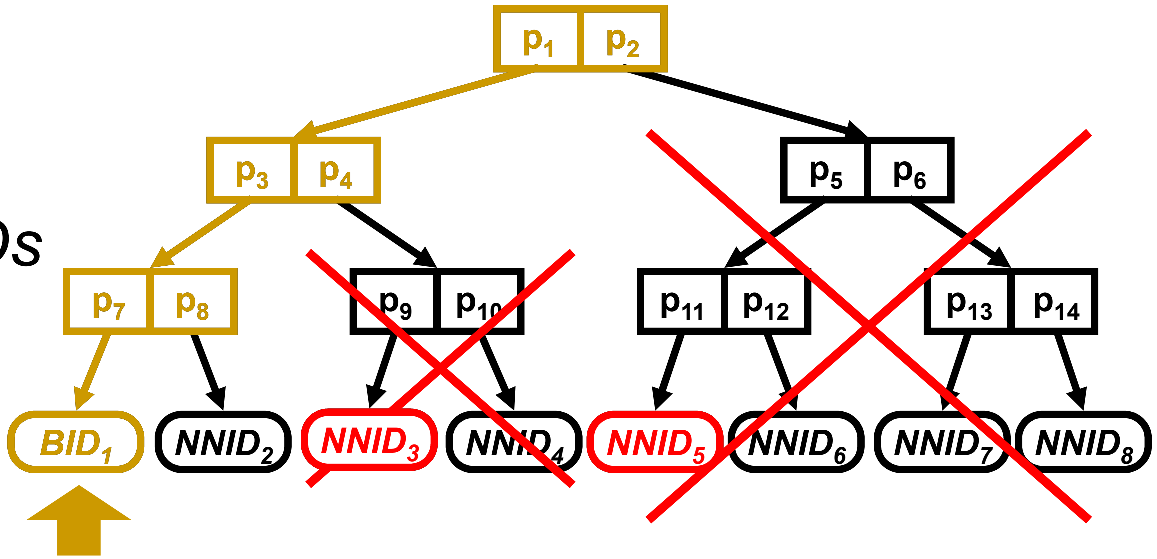
AST: Image Adjustment (cont.)

- The AST is modified on bucket splits and merges
  - Only changed peers are aware of the change (4 and 5)
- When other peer searches
  - Forwards the query to a peer
    - which has a different AST view
  - The incomplete search is detected
    - by too short *BPATH*
  - The search evaluation resumes
    - possibly forwarding the query to some other peers
- Image adjustment is sent back



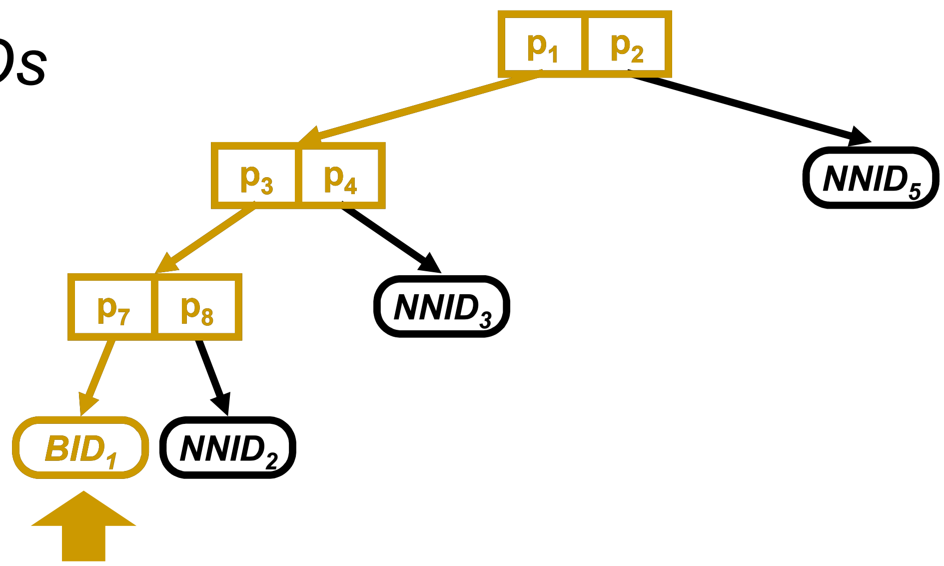
AST: Logarithmic Replication

- The full AST on every peer is space consuming
  - many pivots must be replicated at each peer
- Only a limited AST stored
  - all paths to local buckets
  - nothing more
- Hidden parts
  - replaced by the *NNIDs* of the leftmost peers



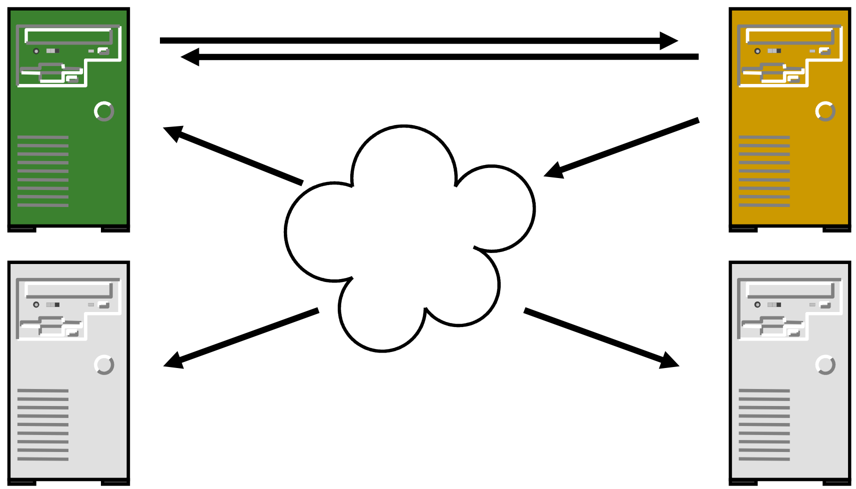
AST: Logarithmic Replication (cont.)

- Result of logarithmic replication
  - The partial AST
- Hidden parts
  - replaced by the *NNIDs* of the leftmost peers



GHT* Joining P2P Network

- A new node joining the network sends “I’m here”
  - Received by each active peer
  - Peers add the node to their lists of available peers
- If a node is needed by a split
  - Get one peer from the list
    - send an activation request
  - The peer sends “I’m being used”
    - the other peers remove it from their lists
  - The peer is “Ready to serve”



GHT* Leaving P2P Network

- Unexpected leaves not handled
  - Requires replication or other fault-tolerant techniques
- Peers without storage
  - Can leave without restrictions
- Peers storing some data
  - Delete all stored data
    - all buckets are merged
  - Reinsert data back to the structure
    - without offering its own storage capacity
- Better leaving/fault-tolerant is a research challenge

Parallel and Distributed Indexes

1. preliminaries
2. processing M-trees with parallel resources
3. scalable and distributed similarity search
4. **performance trials**

Performance Trials

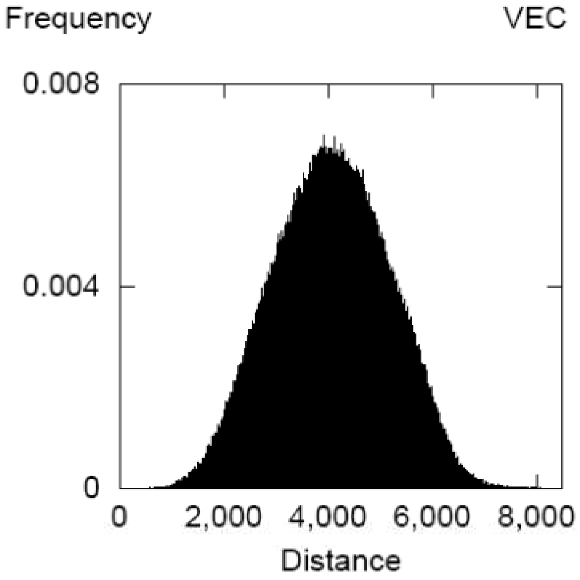
- **Objectives**: show the performance of the distributed similarity search index structure
- The same datasets as for the centralized ones
  - Comparison possible

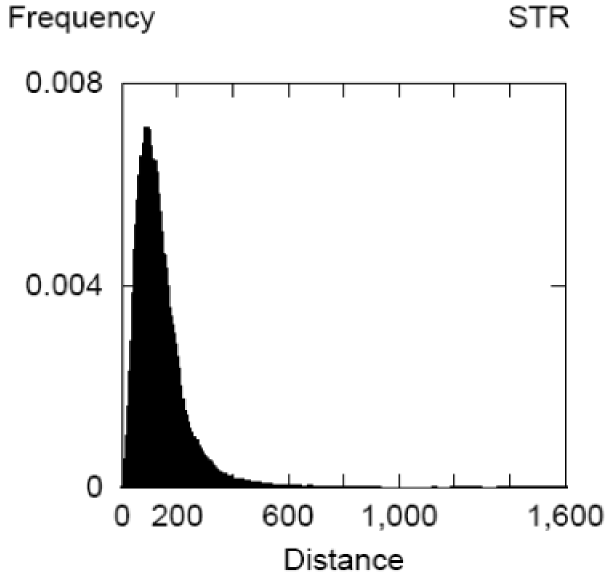
⇒ Experiments show that the response times are nearly constant

Datasets

- Trials performed on two datasets:
  - **VEC:** 45-dimensional vectors of image color features compared by the *quadratic distance* measure
  - **STR:** sentences of a Czech language corpus compared by the *edit distance*

Datasets: Distance Distribution





- Distribution of the distances within the datasets
  - VEC: practically normal distance distribution
  - STR: skewed distribution

Computing Infrastructure

- 300 Intel Pentium workstations
  - Linux operating system
  - available for use to university students
- Connected by a 100Mbps network
  - access times approximately 5ms
- Memory based buckets
  - limited capacity - up to 1,000 objects
- Basic datasets:
  - 100,000 objects
  - 25 peers

Performance Trials: Measures

- Distance computations
  - Number of all evaluations of the metric function
    - either in the AST or in buckets
  - Represent the CPU costs
    - depends on the metric function complexity
      - the evaluation may vary from hundreds of nanoseconds to seconds
- Accessed buckets
  - Number of buckets accessed during a query evaluation
  - Represents the I/O costs

Performance Trials: Measures (cont.)

- Messages sent
  - Transmitted between peers using the computer network
  - Represent the communication costs
    - depends on the size of the sent objects

Performance Trials: Remarks

- Response times are imprecise:
  - not dedicated computers
  - depend on the actual load of used computers and the underlying network
  - other influences

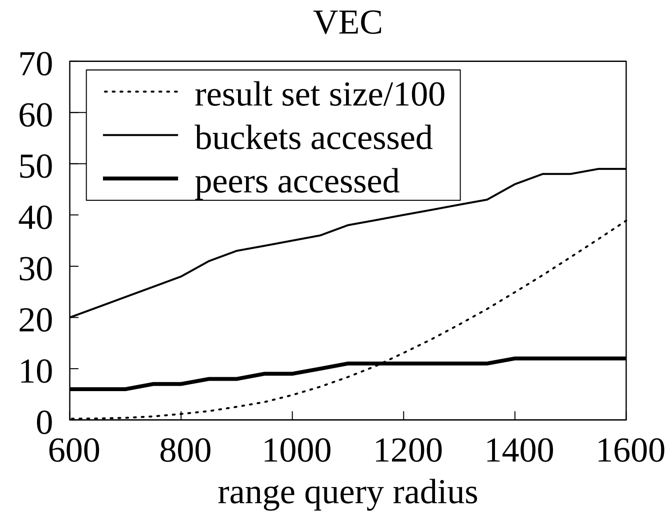
- Query objects follow the dataset distribution
- Average over 50 queries:
  - different query objects
  - the same selectivity (radius or number of nearest neighbors)

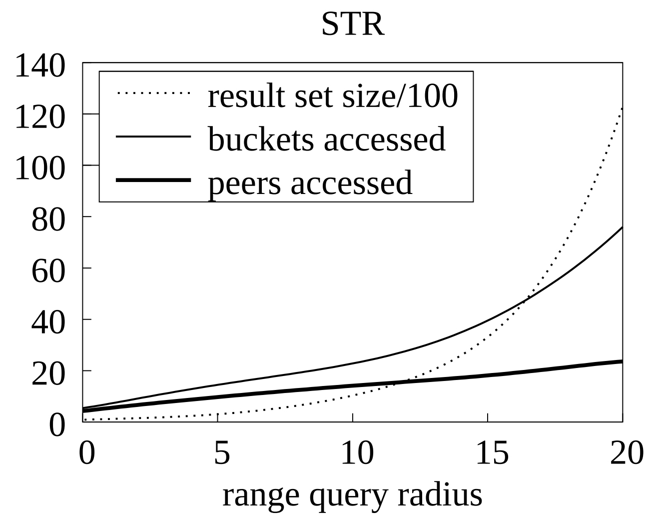
Performance Trials: Outline

- Performance of similarity queries
  - Global costs
    - CPU, I/O and communication
    - similar to the centralized structures
  - Parallel costs
  - Comparison of range and *k*-nearest neighbors queries
- Data volume scalability
  - Costs changes while increasing the size of the data
    - Intraquery parallelism
    - Interquery parallelism

Similarity Queries Global Costs

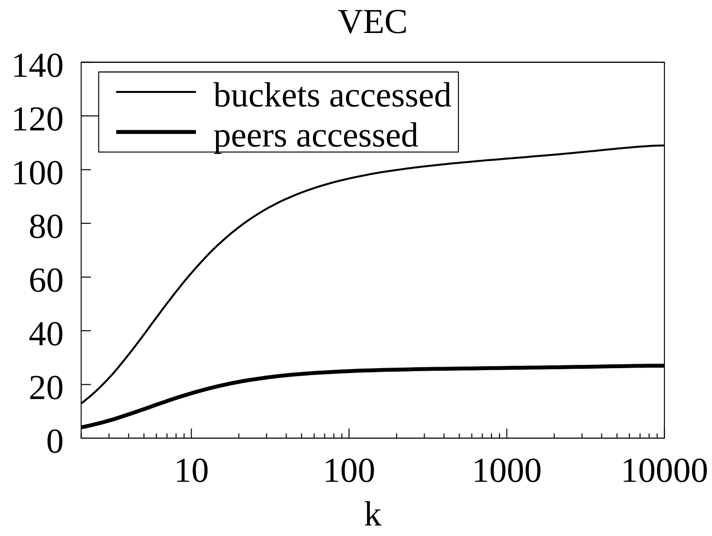
- Changing range query radius
- Result set size
  - grows exponentially
- **Buckets accessed**
  (I/O costs)
  - grows practically linearly
- Similar to centralized structures
- Peers accessed
  - Only slight increase
    - more buckets accessed per peer

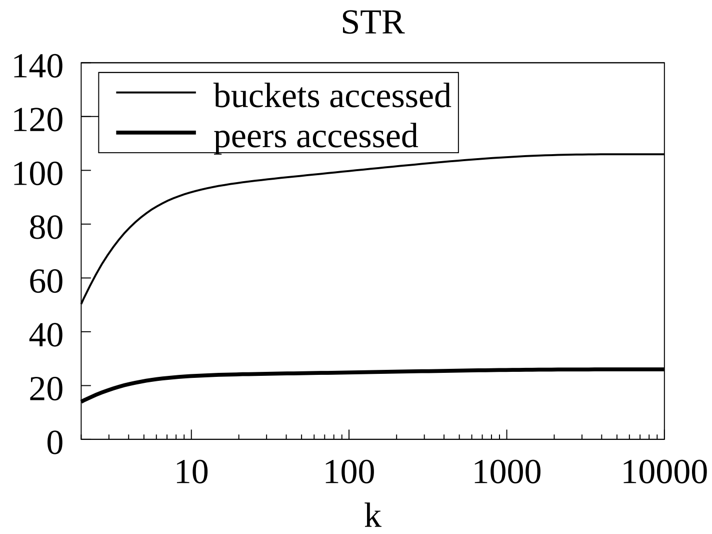




Similarity Queries Global Costs

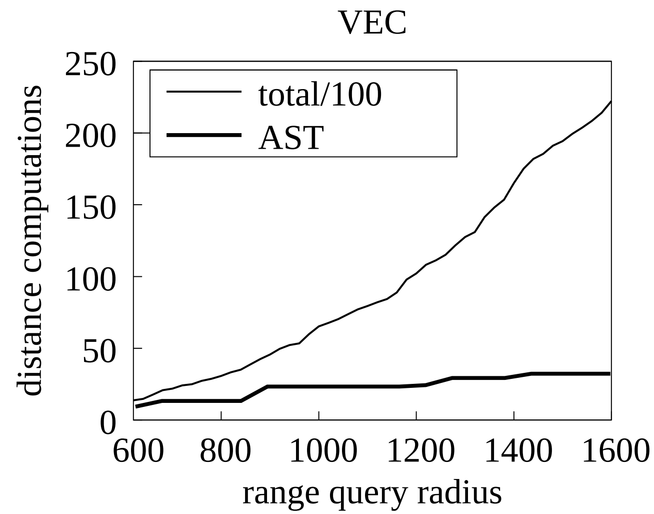
- Changing *k* for *k-NN* queries
  - logarithmic scale
- **Buckets accessed**
  - grows very quickly as *k* increases
- *k-NN* is very expensive
  - similar to centralized structures
- Peers accessed
  - follows the number of buckets
  - practically all buckets per peer are accessed for higher values of *k*

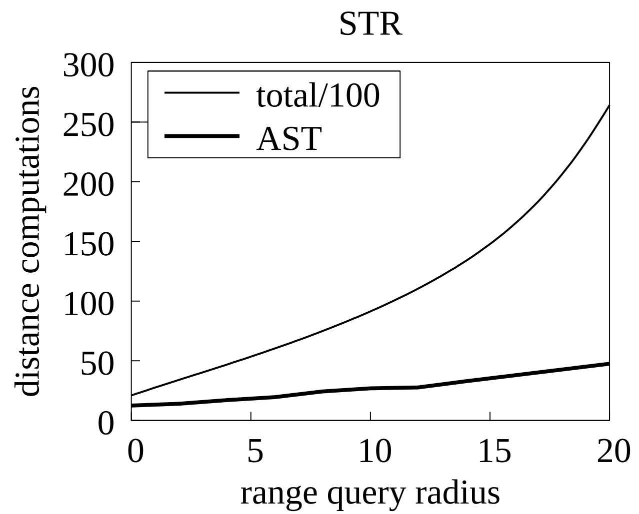




Similarity Queries Global Costs

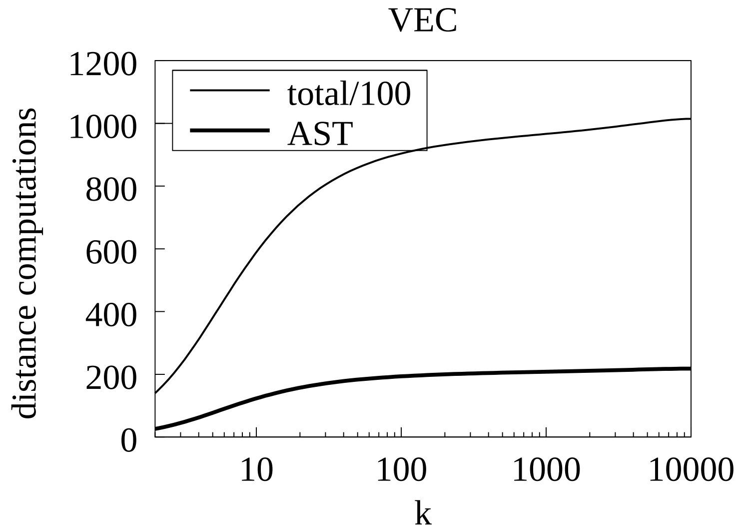
- Changing range query radius
- **Distance computations** (CPU costs)
  - Divided for AST and buckets
    - small percentage of distance comp. during the AST navigation
  - Buckets use linear scan
    - all objects must be accessed
    - no additional pruning technique used
- Similar to centralized structures

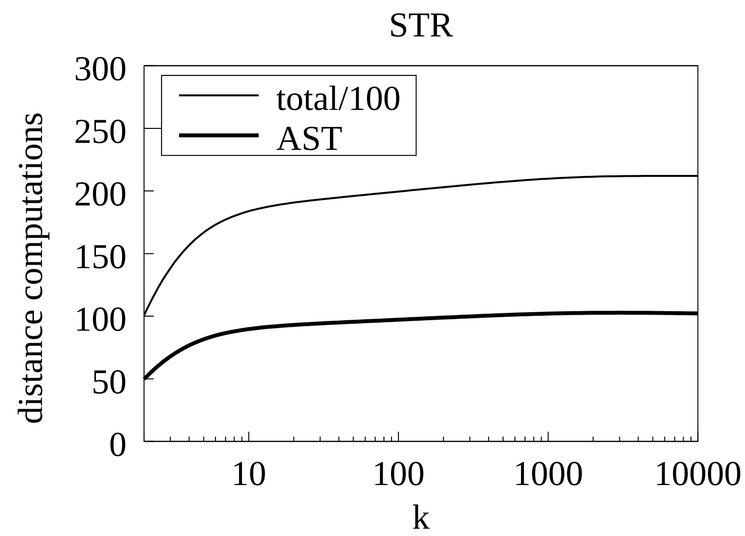




Similarity Queries Global Costs

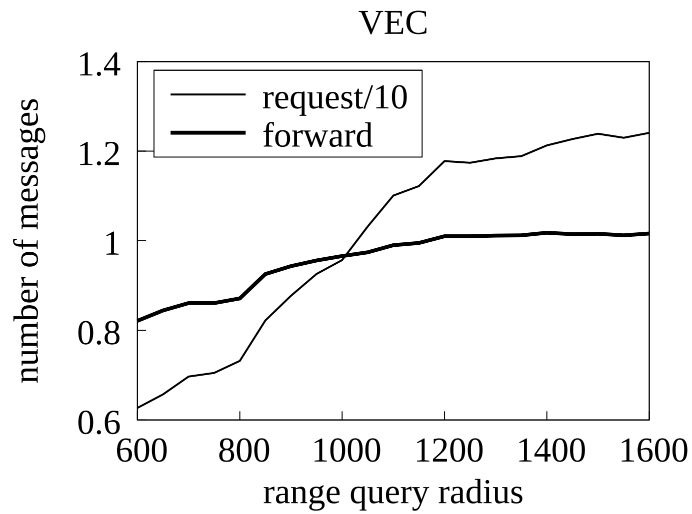
- Changing *k* for *k-NN* queries
  - logarithmic scale
- **Distance computations**
  - only a small percentage of distance computations during the AST navigation is needed
- *k-NN* very expensive
  - also with respect to the CPU costs

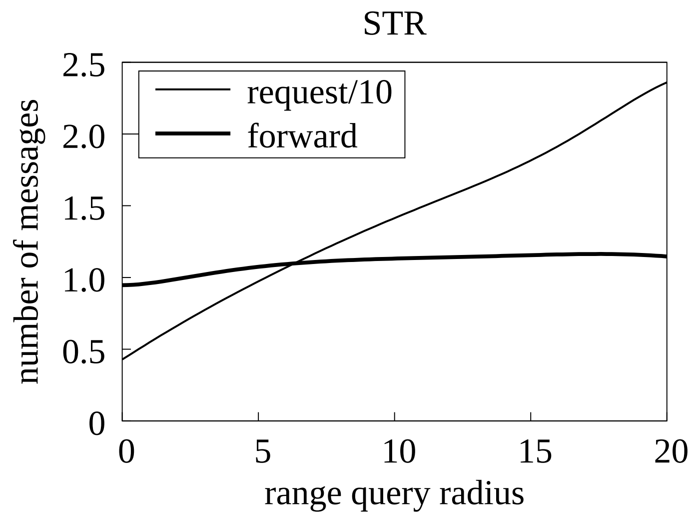




Similarity Queries Global Costs

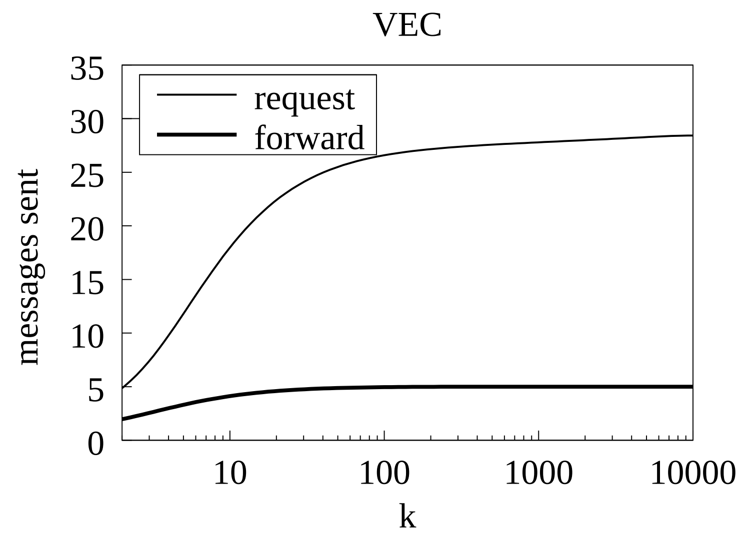
- Changing range query radius
- **Number of messages** (Communication costs)
  - Divided for requests and forwards
    - Forward messages means misaddressing
    - Only 10% messages forwarded
      - even though logarithmic replication used
- No communication in centralized structures

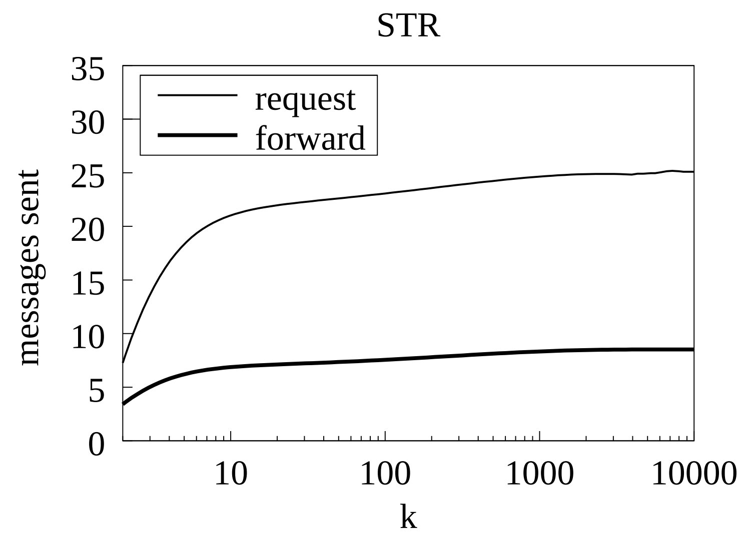




Similarity Queries Global Costs

- Changing *k* for *k-NN* queries
  - logarithmic scale
- **Number of messages**
  - very small number of messages forwarded
  - corresponds with the number of peers accessed
    - practically all peers accessed for *k* greater than 100
  - Slightly higher than for range queries





Similarity Queries Global Costs

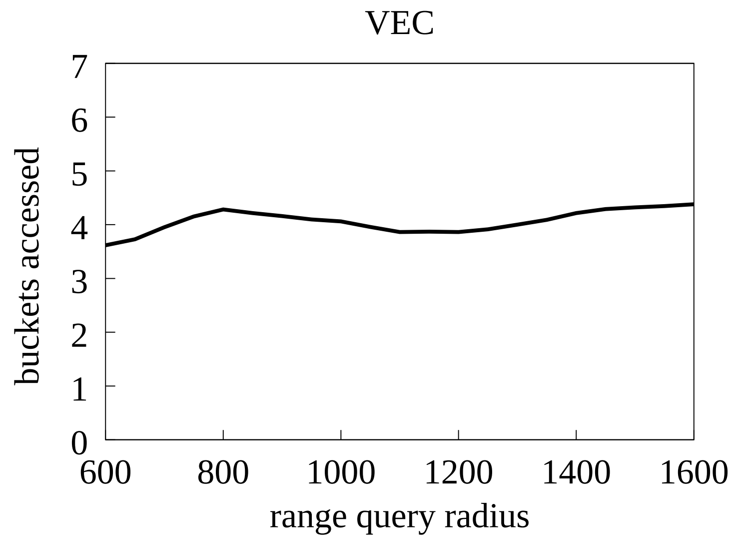
- GHT* is comparable to centralized structures
  - No pruning techniques in buckets
    - slightly increased number of distance computations
  - Buckets accessed on peers
    - not fixed size of blocks, but fixed bucket capacity
- Trends are similar
  - Costs increase linearly

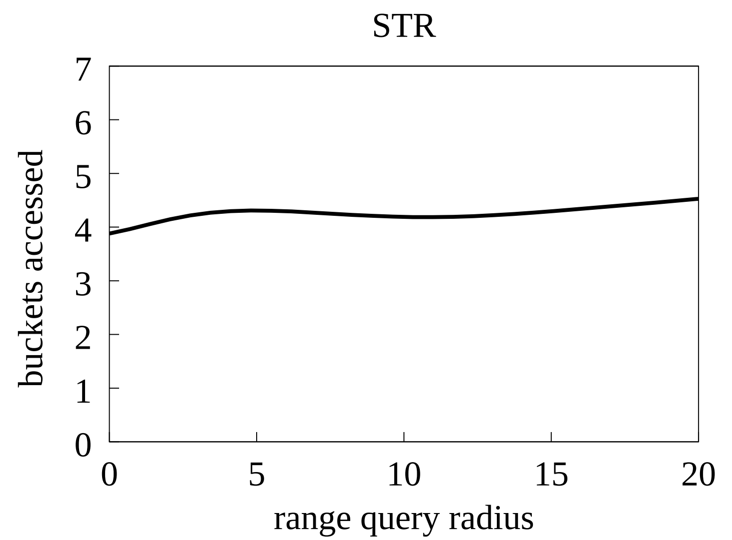
Similarity Queries Parallel Costs

- Correspond to the actual response times
- More difficult to measure
  - Maximum of the serial costs from all accessed peers
  - Example: the parallel distance comp. of a range query
    - number of distance computations at each peer accessed
      - at a peer, it is a sum of costs for accessed buckets
    - maximum of the values needed on active peers
- *k-NN* has the serial phase of locating the first bucket
  - we must sum the first part with the range query costs
  - additional serial iterations may be required if optimistic/pessimistic strategy is used

Similarity Queries Parallel Costs

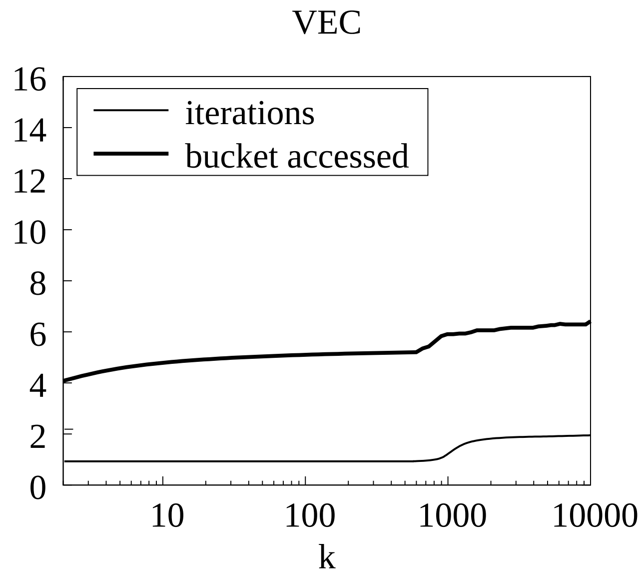
- Changing range query radius
- **Parallel buckets accessed** (I/O costs)
  - Maximal number of buckets accessed per peer
  - It is bounded by the capacity
    - a peer has at most five buckets
- Not affected by the query size

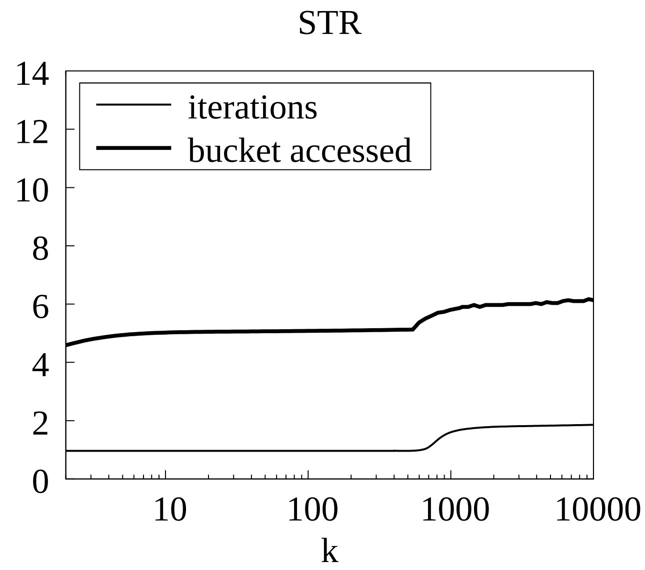




Similarity Queries Parallel Costs

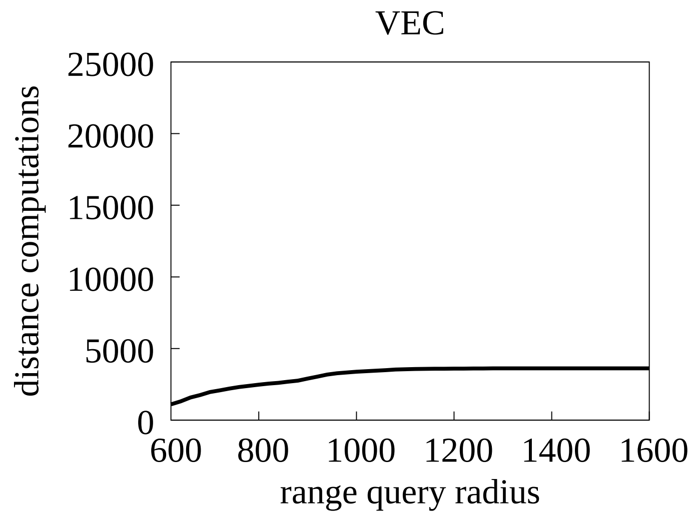
- Changing *k* for *k-NN* queries
  - logarithmic scale
- Iterations
  - one additional optimistic strategy iteration for *k* greater than 1,000
- **Parallel bucket access costs**
  - bounded by the capacity
    - practically all 5 buckets per peer are always accessed
  - second iteration increases the costs

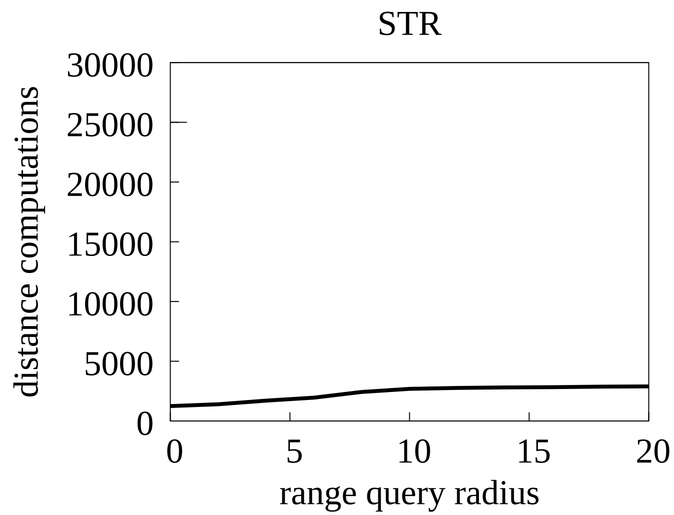




Similarity Queries Parallel Costs

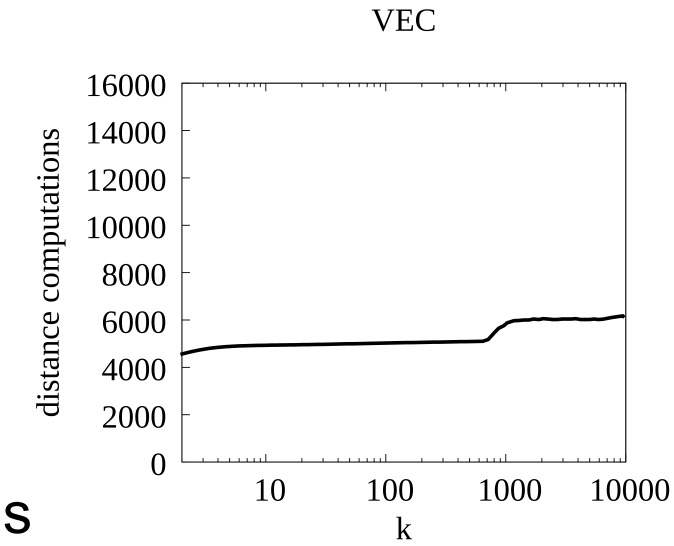
- Changing the range query radius
- **Parallel distance computations** (CPU costs)
  - Maximal number of distance computations per peer
    - the costs of the linear scans of the peer's accessed buckets
  - It is bounded by the capacity
    - a peer has maximally five buckets of maximally 1,000 objects
- Good response even for large radii

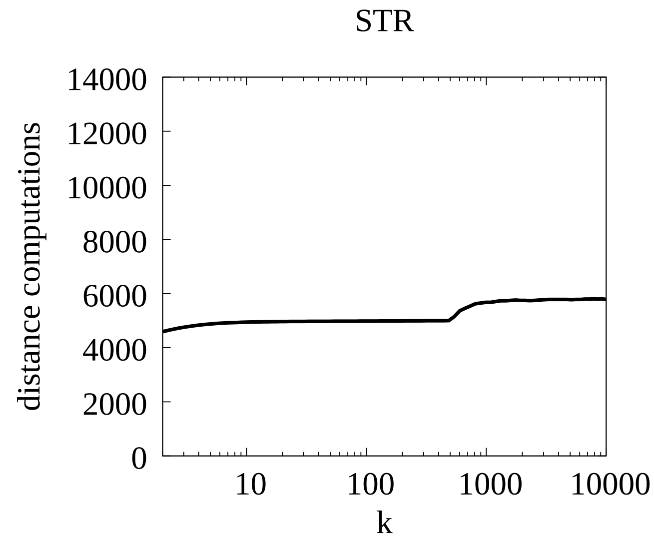




Similarity Queries Parallel Costs

- Changing *k* for *k-NN* queries
  - logarithmic scale
- **Parallel distance computations**
  - bounded by the capacity
    - maximally 5,000 distance computations per peer
    - all objects per peer are evaluated
  - Second iteration (*k* > 1,000)
    - Increases the cost
- Although *k-NN* query is expensive, the CPU costs are bounded



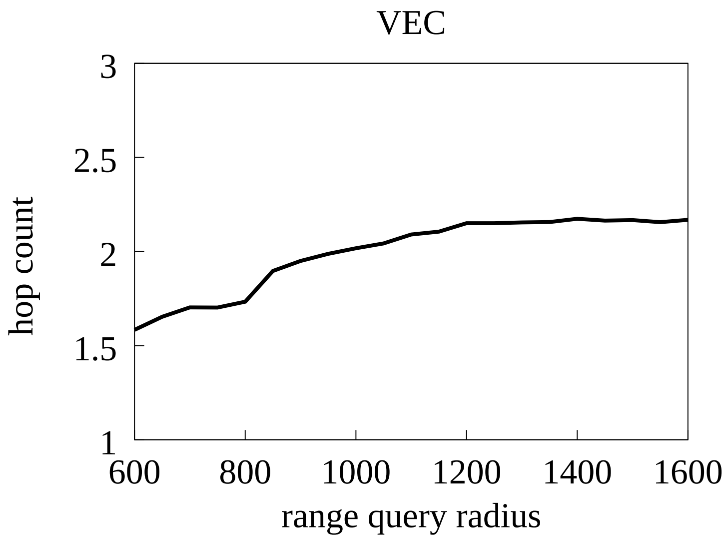


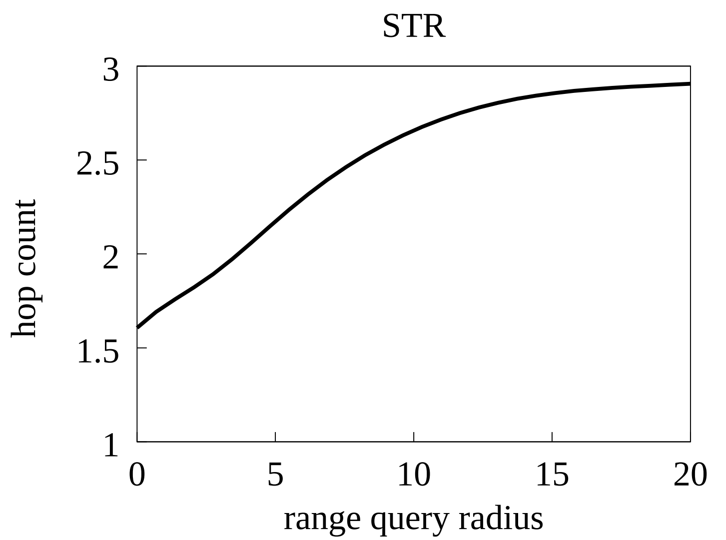
Similarity Queries Parallel Costs

- Measure for the messages sent (the communication costs)
  - during the query execution, the peer may send messages to several other peers
    - the cost is equal to sending only one, because the peer sends them all at once
  - the serial part is thus the forwarding
- The number of peers sequentially contacted
  - hop count

Similarity Queries Parallel Costs

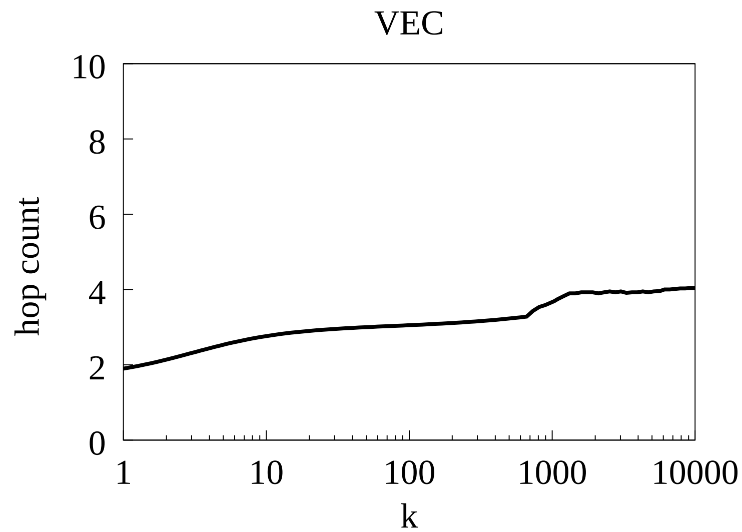
- Changing range query radius
- **Hop count**
  (Communication costs)
  - logarithmically proportional to the number of peers accessed
  - in practice, this cost is very hard to notice
    - forwarding is executed before the local buckets scan

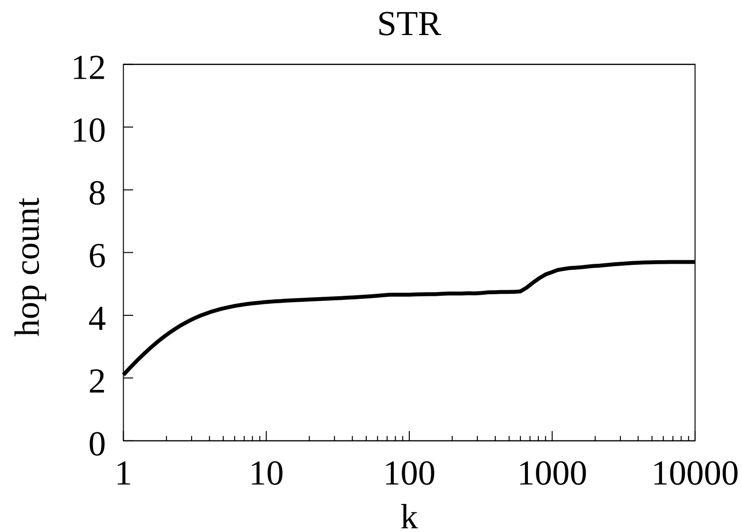




Similarity Queries Parallel Costs

- Changing *k* for *k-NN* queries
  - logarithmic scale
- **Hop count**
  - Since only few messages are forwarded, the *k-NN* queries have practically the same costs as the range queries
  - Small amount of additional hops during the second phase
    - approximately one additional hop is needed

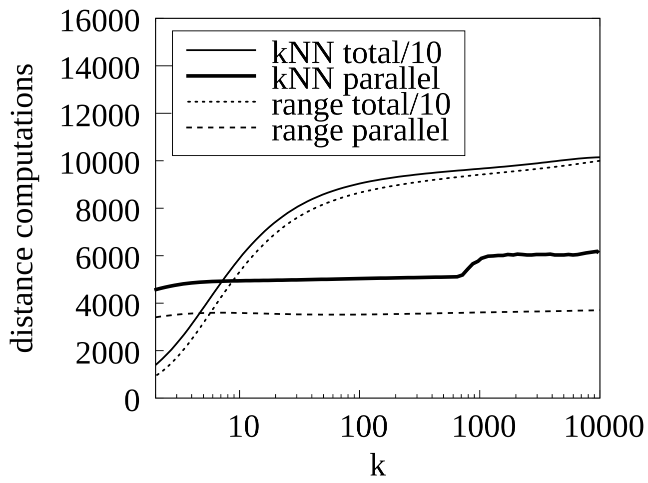




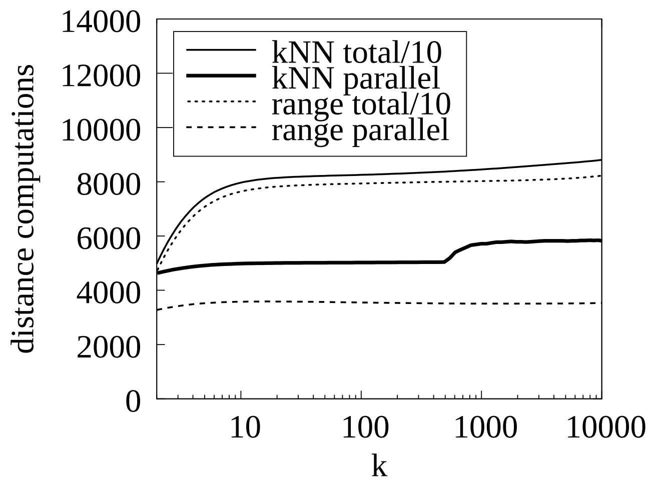
Similarity Queries Comparison

- *k-NN* and range queries
  - logarithmic scale
  - range query has the radius set to the distance of the *k*-th nearest object
    - that is the perfect estimate
- **Total distance computations**
  - the *k-NN* query is slightly more expensive than the range query
- **Parallel distance computations**
  - clearly visible differences of the first phase and additional iteration(s)

VEC



STR



Similarity Queries Parallel Costs

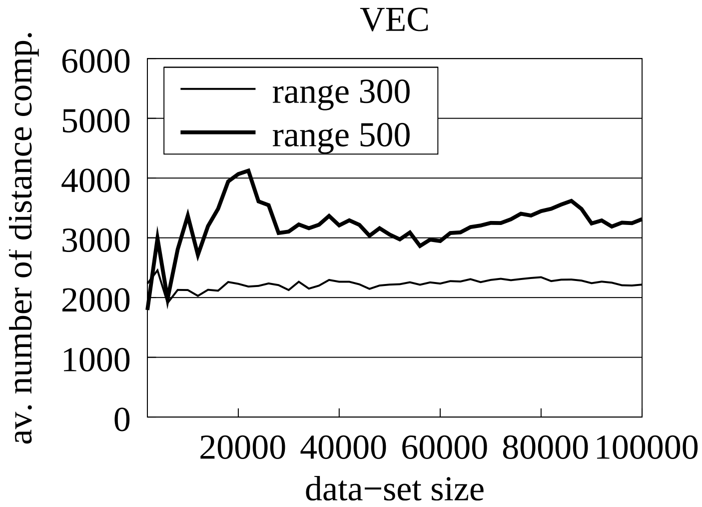
- GHT* real costs summary
  - the real response of the indexing system
- GHT* exhibits
  - constant parallel CPU costs
    - distance computations bounded by bucket capacity
  - Constant parallel I/O costs
    - number of buckets accessed bounded by peer capacity
  - Logarithmic parallel communication costs
    - even with the logarithmic replication

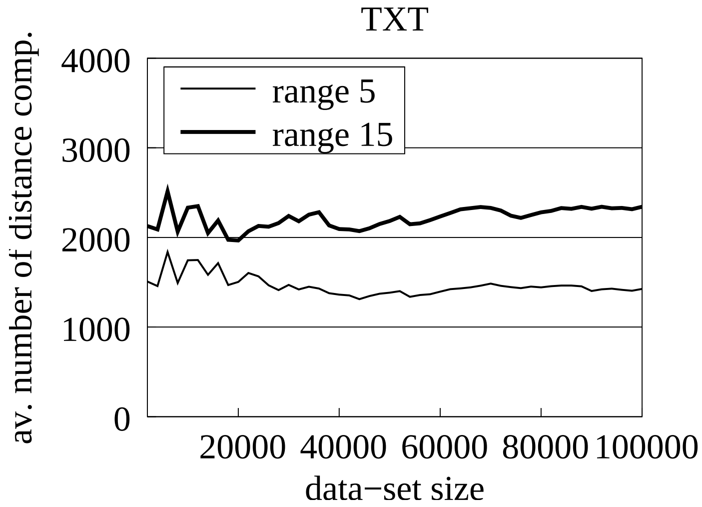
Data volume scalability

- Dataset gradually expanded to 1,000,000 objects
  - measurements after every increment of 2,000 objects
- Intraquery parallelism
  - parallel response of a query measured in distance comp.
  - maximum of costs incurred at peers involved in the query
- Interquery parallelism
  - simplified by the ratio of the number of peers involved in a query to the total number of peers
  - the lower the ratio, the higher the chances for other queries to be executed in parallel

Data volume scalability

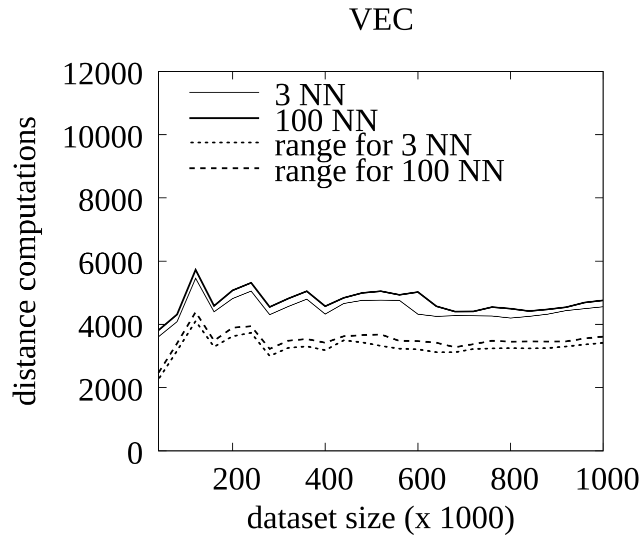
- Changing dataset size
  - two different query radii
- **Intraquery parallelism**
  - Practically constant responses
    - even for the growing dataset
    - some irregularities for small datasets observed
  - Larger radii result in higher costs
    - though, not much

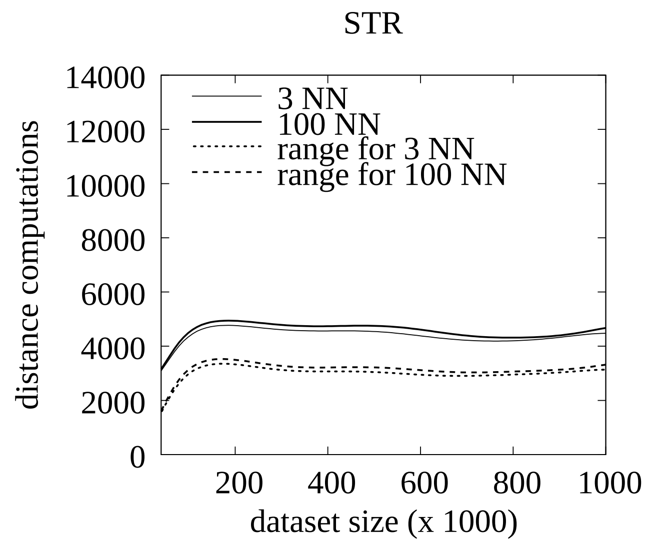




Data volume scalability

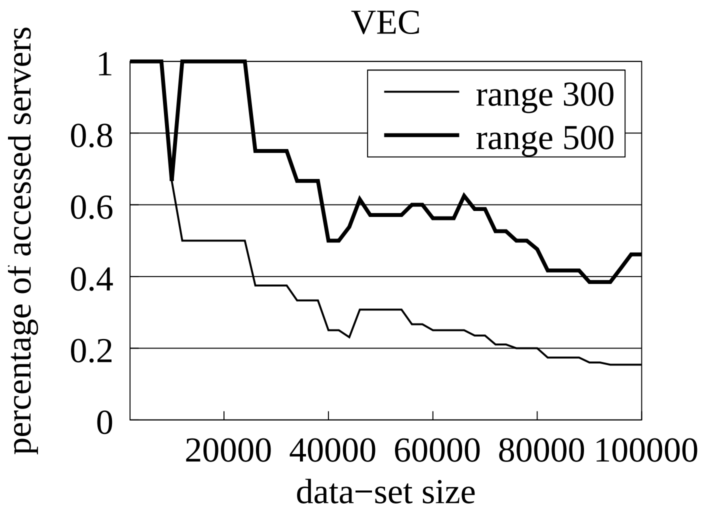
- Changing dataset size
  - two different *k* for *k-NN*
  - corresponding range queries
- **Intraquery parallelism**
  - by analogy to range queries the responses are nearly constant
  - There is a small difference for different values of *k*

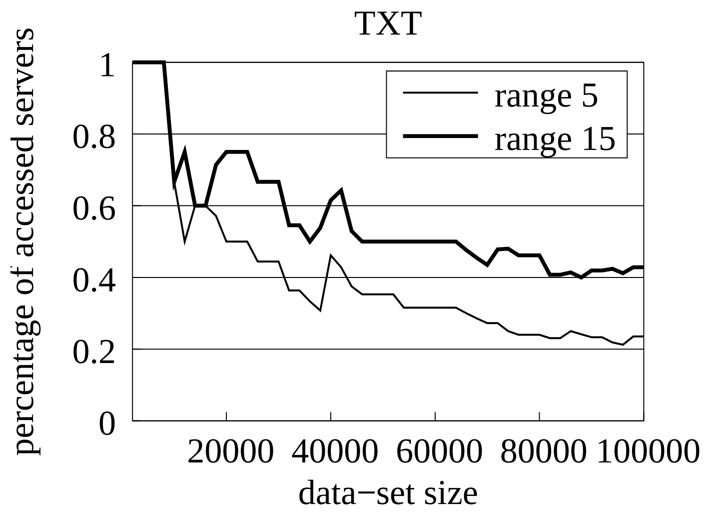




Data volume scalability

- Changing dataset size
  - Two different query radii
- **Interquery parallelism**
  - As the size of the dataset increases, the interquery parallelism gets better
  - Better for the smaller radii
    - smaller percentage of peers involved in a query





Data volume scalability

- GHT* scalability for one query
  - Intraquery parallelism
    - both the AST navigation and the bucket search
  - Remains practically constant for growing datasets
- GHT* scalability for multiple queries
  - Interquery parallelism
    - a simplification by percentage of used peers
  - Allows more queries executed at the same time as the dataset grows