Complex Queries in DHT-based Peer-to-Peer Networks

Matthew Harren et al.

Ali Taleghani

March 30, 2005

Outline

- Introduction
- Motivation
- Why not P2P Databases
- Substring Search
- P2P Query Processing
- CANDy
- Range Queries
- Conclusion & Comments

Introduction

- P2P storage networks
  - Decentralized control
  - Self-organization
  - Large amount of data
  - Distributed systems – each node has identical capabilities
  - Single point of failure is eliminated

Motivation

- Three basic search methods
  - Centralized directory
    - Single point of failure
    - Scalability issues
    - Allows for powerful queries
  - ID-only access
    - DHTs can be used - fast
    - Exact match queries are possible
  - Flooding query
    - Different queries can be implemented
    - Expensive in terms of network bandwidth

Motivation

- Complex query facilities over DHTs
- Preserve the DHT efficiency
- Want join, selection, grouping etc.
- Current paper provides direction of research

Why not P2P Databases

- Do not want perfect storage semantics and carefully administered data
  - Ease of use, scalability, robustness to volatility
- Users do not want to deal with DB
- All data seen relational at some level
  - Can hide that level from user
- Want P2P query processing, separate it from problem of storage

Textual Similarity Search

- Based on hash indexes
- Split string to be indexed into “n-grams”
  - “Beethoven” split into Bee, eet, eth, tho, hov, ove, ven
- Hash each n-gram and build index
- Given substring, split into n-grams
  - “thoven” split into tho, hov, ove, ven

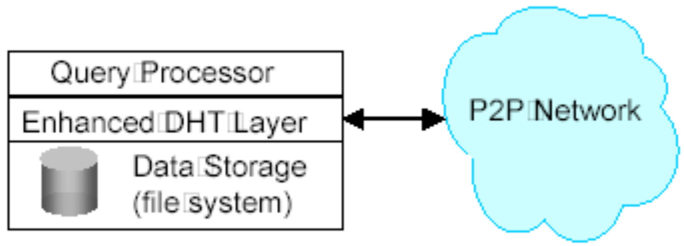
Textual Similarity Search

- Look up index for each n-gram in query
- Results are grouped by file ID
  - “Beethoven” in our case
- For strict substring:
  - # of same fileID must be equal to # of n-grams in query
  - For “Beethoven” and “thoven” it’s four
- Some post processing is still necessary

P2P Query Processing

- Broad Applicability
  - Interact with user’s file-systems as existing P2P systems
- Minimal Extension to DHT APIs
  - No complication to current DHTs
  - Need query processing that is portable across multiple DHT implementations

Architecture



- Three layers
  - Data storage (File System)
  - Enhanced DHT layer (with networking)
  - Query Processor

Data Store

- File-system or wrapper over database
- Iterator
  - Scan through the set of objects
- Accessors to attributes of objects
- Metadata interface for objects
- Accessors to additional attributes

DTH Layer

- Iterator called *lscan*
  - Scans through all DHT entries on a machine
- Callback *newData*
  - Notifies higher layers when new data added
  - Used to deal with insertions in timely manner
  - Used for temporary tables

Query Processor

- Parallel implementation of query operators
- Specifying queries & iterating through results
- Support two query APIs:
  - Graph-scripting for specifying explicit query plans
  - Simplified SQL interface for declarative queries

Namespace

- DHT assumes flat identifier space
- Complex queries need to name tables, tuples and fields
- Can implement hierarchical namespace
  - Partition identifier in multiple fields
  - Each field identifies objects of same granularity

Query Processing Operators

- Join relation *R* and *S*
  - Query node initializes temp DHT namespace T_{joinID}
  - Node receives data from *R*
  - Join attribute extracted & inserted into T_{joinID}
  - newData calles QP layer to check local data for matches
  - Matches pipelined to next iterator in plan

Status of Project

- Join operation implemented using CAN
- Different Join variants implemented
  - Each node performs join
- Hotspots in all dimensions discovered
  - Storage
  - Processing
  - Routing

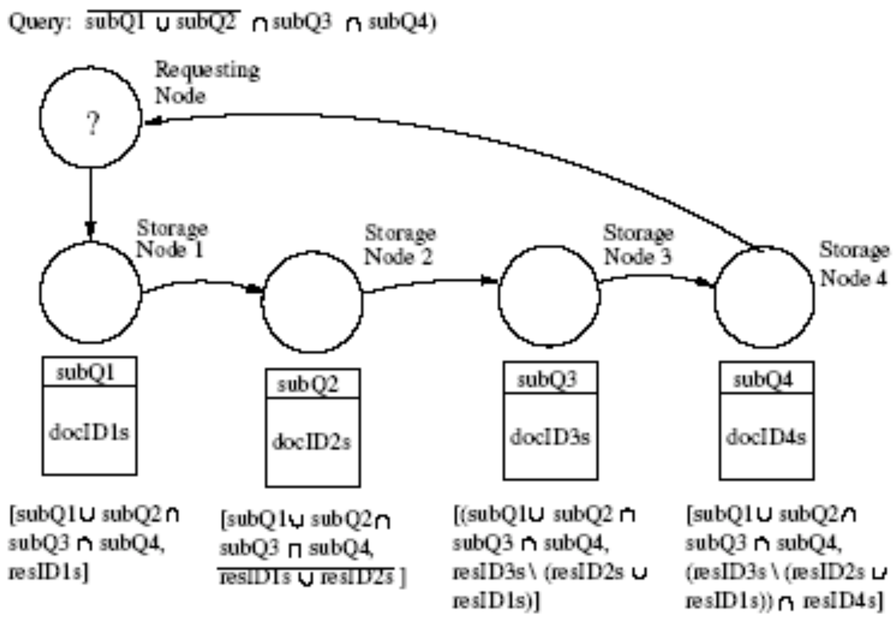
CANDy

- Content Addressable Network Directory
- Use two DHTs:
  - Index DHT
    - Stores property & index information
    - Many resources can have same property
  - Resource DHT
    - Stores actual pointer to resources

CANDy Query Processing

- User agent identifies properties in query
  - Using property descriptors
- Each property represents set of resIDs
- Query translated into sequence of set operations on resID sets
- Query sent to storage nodes – handle sub-query
- Last node returns results (resIDs) to user agent

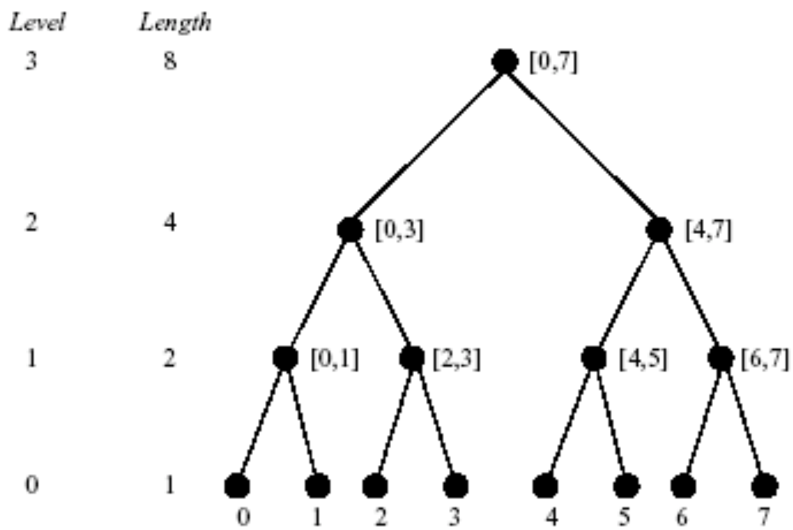
CANDy Query Processing

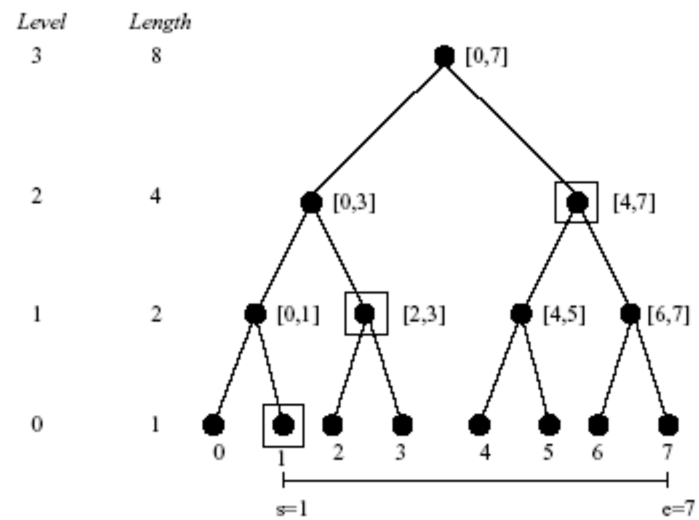


Range Queries

- Based on Range Search Trees (RST)
- RST is complete & balance binary tree
- Each node represents different range in system
- Each leaf node corresponds to single value in system
- Each non-leaf corresponds to union of two children’s ranges
- Range query decomposed into O(log Rq) sub-queries (Rq is range length)

Range Search Tree





Range [1-7] broken down into 3 sub-ranges

Conclusion

- A Framework for queries over DHTs suggested
- Three layered architecture to handle complex queries
- Join algorithm explained
  - Implemented in CAN

Comments

- Join algorithm explained in little detail
  - Network overhead
- Many details missing
  - 3 Layer architecture not a real contribution
  - No other operators explained
- The problem of identifier exists in all approaches
- None handles *ALL* complex queries

J. Gao, P. Steenkiste, (CMU), "Efficient Support for Range Queries in DHT-based Systems" (CMU technical report)

D. Bauer, P. Hurley, R.Pletka and M. Waldvogel. Bringing Efficient Advanced Queries to Distributed Hash Tables. In *Proceedings of IEEE LCN*, November, 2004

M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker and I. Stoica. Complex Queries in DHT-based Peer-to-Peer Networks, In *Proc. 1st Int. Workshop on Peer-to-Peer Systems* (IPTPS), 2002.