Lecture 4
Query Processing in P2P
(Part A)

Zhou Shuigeng

April 6, 2007
Outline

- Introduction
- Approximate range selection in P2P
  - A. Gupta, D. Agrawal, and A. E. Abbadi
- A Peer-to-peer Framework for Caching Range Queries
  - O. Sahin, A. Gupta, D. Agrawal, A. E. Abbadi
- One Torus to Rule Them All: Multi-dimensional Queries in P2P Systems
  - P. Ganesan, B. Yang, and H. Garcia-Molina
Introduction
Query processing in P2P environment is related to peers’ characteristics, mainly:

- **Topology**: how peers are connected to each other.

- **Data Placement**: how data or metadata is distributed across the network of peers.

- **Message routing**: how messages are propagated through the network.
System Requirements

- Peers’ characteristics in P2P depend on query processing system requirements
  - Expressiveness
    - To what detail that the query language can describe the desired data
  - Comprehensiveness
    - How many results the system can return, partial or complete?
  - Autonomy
Evaluation Metrics

- **Efficiency**
  - Resources consumed for query processing
    - bandwidth, processing power, storage, etc.

- **Quality of service**
  - Number of results
  - Response time

- **Robustness**
  - Whether the results are stable in the presence of local system failures, peers leaving/entering
Expressiveness (1)

- **Key lookup**
  - Most widely used query form in data sharing P2P systems
  - How to improve efficiency & robustness further?

- **Keyword searching**
  - Used for data sharing and IR in P2P
  - Keyword join

- **Similarity based query**
  - Top-k searching, skyline
Expressiveness (2)

- Aggregates
  - Approximate processing
  - There is some work on simple COUNT and RANGE query
  - How about MAX, MIN, SUM, MEDIAN, etc.?

- SQL
  - Current research on supporting SQL in P2P systems is very preliminary
  - PIER project supports a subset of SQL over a P2P framework
Expressiveness (3)

- On-line analytical processing (OLAP)
  - One research paper appeared

- Data mining
  - Not much work has been done
    - Association rule mining
    - clustering

- ......
Major Challenges

- No global schema & metadata
- Dynamic & ad hoc network structure
- Peers’ autonomy
- ......
Open Problems

- Query processing models
  - Blind vs. heuristic searching
  - Data warehousing: using views
  - Schema mediation
  - Some others (?)

- Query optimization
  - ?????

- Query language (?)
Related Techniques

- Network technology
- P2P data sharing systems
- Distributed information retrieval
- Data Warehousing & OLAP
- Data Mining & KDD (?)
- Heterogeneous data sources integration
- Distributed database query processing
- ......
Approximate Range Selection Queries in Peer-to-Peer Systems
(From CIDR 2003)

A. Gupta, D. Agrawal, and A. E. Abbadi

University of California Santa Barbara
About the Paper

- Presents an architecture for a P2P system that shares data in the form of relations
  - Peers cache horizontal partitions of relations
  - Queries submitted in the form of SQL statement
  - Using Locality Sensitive Hashing to locate partitions of relations relevant to the query
Motivations

- **Selection** is a fundamental operation to retrieve data from the database.
- P2P users often ask broad queries even when they are only interested in a few results, and therefore do not expect perfect answers.
Contributions

- An architecture for a relational data sharing peer-to-peer system
- A hashing based mechanism to locate data partitions relevant to a query
- This paper may constitute an initial step to enable general query processing over P2P data sharing architectures
System Scenario (1)

- Peers share relational data
  - In the form of database tuples and relations
  - A global schema is known to all the peers in the system
  - Sources of data are part of the peer-to-peer system (i.e., they are also peers in the system), and are known to all the peers
  - Peers are allowed to cache horizontal partitions of relations
  - Peers submit SQL queries to the system
Selection on a relation can be only on one attribute at a time

When a query is submitted

- The query is converted into a plan where all the selects are moved toward the leaves as much as possible
- Peer locates relevant relation partitions in the system that can help answer the query
- The located peers caching relevant partitions send the data over to the requesting peer
An Example (1)

- The global schema
  - **Patient**\(\text{patient id, name, age}\)
  - **Diagnosis**\(\text{patient_id, diagnosis, physician_id, prescription_id}\)
  - **Physician**\(\text{physician_id, name, age, specialization}\)
  - **Prescription**\(\text{prescription_id, date, prescription, comments}\)
The query is like this:

to find out what prescriptions have been provided to patients diagnosed with *Glaucoma* and with age in 30 to 50 between Jan 2000 and Dec 2002.

In SQL:

```
Select Prescription.prescription from Patient, Diagnosis, Prescription
where 30 <= age <= 50
and diagnosis = "Glaucoma"
and Patient.patient_id = Diagnosis.patient_id
and 01 - 01 - 2000 <= date <= 12 - 31 - 2002
and Diagnosis.prescription_id
= Prescription.prescription_id.
```
An Example (3)

Query processing plan

Advanced Distributed Computing

21
An Example (4)

Data retrieval

Advanced Distributed Computing
Locating Relevant Partitions

- Problem statement
  - Given a relation R and the selection range \((\text{start}, \text{end})\) over an attribute of the relation, find out if there is a peer that caches a partition of relation R that can help us compute the desired selection.
  - The hard problem is to find the peers containing the desirable partitions in a P2P systems.
Locality Sensitive Hashing

- Definition: If $A$ and $B$ are two sets of values from domain $D$ then a family of hash functions $H$ is said to be locality preserving if for all $h \in H$ we have
  \[
  \Pr[h(A) = h(B)] = \text{sim}(A, B)
  \]
- For range sets $R$ and $Q$, using the Jaccard set similarity measure
  \[
  \text{sim}(Q, R) = \frac{|Q \cap R|}{|Q \cup R|}
  \]
Min-wise Independent Permutations

Given a domain $D$, consider a random permutation $\pi$ of $D$. Assume that the elements of $D$ are totally ordered. Given a range set $Q \subseteq D$, the hash function $h_\pi$ is defined as

$$h_\pi(Q) = \min\{\pi(Q)\}$$

this hash function family satisfies the following property

$$Pr_\pi[h_\pi(Q) = h_\pi(R)] = \frac{|Q \cap R|}{|Q \cup R|}$$
We say that $F \subseteq S_n$ is min-wise independent if for any set $X \subseteq [n]$ and any $x \in X$, when $\pi$ is chosen at random in $F$ we have

$$\Pr(\min\{\pi(X)\} = \pi(x)) = \frac{1}{|X|}$$

all the elements of any fixed set $X$ have an equal chance to become the minimum element of the image of $X$ under $\pi$. 
Permutation Operation (1)

- Assume we are dealing with sets of 8-bit integers
  - Take an 8-bit key that has exactly 4 random bits set to 1
  - For an integer in the set, move the bits corresponding to the position of 1’s in the key to upper half and the others to the lower half in order
  - Next, choose a 4-bit key with exactly 2 random bits set to 1. Again permute the bits in the 8-bit integer using this key for each of the 4-bit halves
And so on, until each pair of 2 bits has been permuted.

The keys for this permutation function are representable as three 8-bit integers. The permutation operation produces an integer as the final output.

The hash function $h_{\pi}(Q)$ applies the above permutation operation on each integer in the range $Q$ and then takes the minimum of the resulting integers.
An Example

(a) first iteration

(b) next iteration

(c) last iteration
The query range is hashed to the identifier space using an appropriate locality sensitive hash (LSH) family. Peers in the system are also mapped to the same identifier space using any randomly distributed hash function (e.g. SHA-1 [3]). Chord is used to map data partition identifiers to peer node identifiers, and provide the lookup and routing facility.
System Architecture (2)

Structure of distributed hash table
Discussions

- Each attribute corresponds to a distributed hash table, space cost is a problem for large scale databases.
- How to construct LSH for float, string and other date types?
- How to initialize the cached relations and tuples? Based on to history results?
- Can the method be applied to more complex queries processing?
A Peer-to-peer Framework for Caching Range Queries

(From ICDE 2004)
Ozgur D. Sahin, Abhishek Gupta
Divyakant Agrawal, Amr El Abbadi

Department of Computer Science
University of California at Santa Barbara
Outline

- Motivation
- Range mapping
- System overview
- Experimental results
- Conclusion and future work
Motivation

- All queries are answered by the server
  - Server is overloaded
  - Scalability, availability
- Same/similar queries are evaluated multiple times
Motivation

- Users share their cached answers
- Server is contacted only if the P2P layer cannot find an answer
P2P Systems

- File sharing: Napster, Gnutella, KaZaA, ...
  - Central index or flooding
- Structured P2P systems: CAN, Chord, Pastry, Tapestry, ...
  - Distributed Hash Tables (DHTs)
  - Efficient Routing: logarithmic/sublinear
**CAN**

- Uses a $d$-dimensional virtual space for routing and object location
- Virtual space is partitioned into zones and each zone is maintained by a peer
- Every peer is responsible for the objects that are hashed into its zone

2-dimensional CAN
Extending DHT functionality

- DHTs are designed for exact-match queries
- Piazza [Univ. of Washington], Hyperion [Univ. of Toronto], PIER [UC Berkeley]
- Extend DHTs for supporting range queries
  - Selection of ranges is a primary operation for any kind of data analysis
- **Main Goal:** Utilize a DHT in order to materialize and locate cached answers of range queries
Range Queries

- Given a range query, find the cached answers that can be used to compute the query answer
  - Example: If the result of \( <20,35> \) is already cached in the system, then the query \( <22,30> \) can be answered using the cached result
  - \( <22,30> \) is subsumed by \( <20,35> \); so the cached result is the super-set of the answer
DHTs for locating ranges

- Can we use original DHTs?

  Use range string as key:

  Query: \(<22,30>\)
  Hash string: “\(<22,30>\)”

  Finds exact answers but not the similar ones!

- Range Queries over DHTs: Approximate answers
  [Gupta et al.], Data partitioning [Andrzejak et al.]
Extending CAN

- For single attribute, the virtual space is a 2-dimensional CAN
- The boundaries are determined by the domain of the range attribute

Virtual space when attribute domain is [20,80]
Mapping Scheme

- Range \( <x, y> \) is mapped to point \( (x, y) \)

- Super-ranges are only in the upper-left region

\[
\begin{align*}
\langle 40, 60 \rangle & \quad \rightarrow \quad (40, 60) \\
\end{align*}
\]
Space Partitioning

- Virtual space is partitioned into rectangular zones
- Each zone is assigned to an active peer
- With this mapping, the data source is responsible for the top-left zone
Space Partitioning

- Active/Passive peers

- Active Peers
- Passive Peers

Data Source
Space Partitioning

- Active/Passive peers

Active Peers
Passive Peers
Space Partitioning

- Active/Passive peers

![Diagram of Active and Passive Peers]
Space Partitioning

- **Active/Passive peers**

  - Each active peer keeps a list of passive peers
  - Passive peers register with active peers
An active peer splits its zone when it is overloaded
- Load can be due to storage or bandwidth, etc.

Split line is selected by the owner of the zone
- Even partitioning of the zone and the cached results

New zone is assigned to a passive peer
Routing

- Same as in CAN (Greedy routing)
- Each zone passes the message to the neighbor closest to the destination

Query: <50,55>
Forwarding

If no result is found at the destination, then...

- The zones on the upper-left region may have super-ranges
- Destination zone forwards the request to upper-left zones

![Diagram showing forwarding process with points (50, 55) and arrows indicating direction of request.]
Acceptable Fit

- How far to forward?

- Forwarding is controlled by a parameter: `AcceptableFit`

- It is a real value between [0,1]:
  
  \[ \text{offset} = \text{AcceptableFit} \times |\text{domain}| \]

- Acceptable range for a range query \(<\text{low, high}>\) is then:
  
  \(<\text{low} - \text{offset}, \text{high} + \text{offset}>\)
Forwarding Schemes

- Two schemes for forwarding:
  - **Flooding**: Flood to all candidate zones
  - **Directed Forwarding**: Iteratively forward to a single neighbor, that has the largest overlap with the acceptable region
    - Stop if a result is found or a certain number of peers are contacted (*DirectedLimit*)
Flooding vs. Directed Forwarding

Flooding

Directed Forwarding
\((Directed\ Limit=2)\)
Updates

- Tuple with value 40 is updated!

Go to the corresponding point, (40,40), and flood to the upper-left region

- Costly, so we need better solutions
  - Batching updates
Multiple range attributes

- Each attribute maps to two dimensions
- A range query over $k$ attributes is mapped to a point in $2^k$-dimensional CAN

$$(20<A<40, 50<B<60)$$

Forwarding
- Decreasing coordinates along odd dimensions
- Increasing coordinates along even dimensions

$$(10<A<50, 40<B<70) \Rightarrow (10,50,40,70)$$
Experiment Settings

- Single attribute with domain [0, 500]
- The system is initially empty
- Range queries are selected uniformly at random
- For every zone:
  \[ \text{Split Point}=5, \text{Routing Threshold}=3 \]
Flooding vs. Directed Forwarding

Performance with flood forwarding

Performance with directed forwarding
Routing is scalable

Visited zones with Flood forwarding

Visited zones with Directed forwarding
Load Distribution

No forwarding (AF=0)  
DL=4 and AF=1

1000 peers, 10000 queries
Conclusion and Future Work

- **Conclusion**
  - We presented a simple yet powerful mapping for ranges which allows us to leverage DHT infrastructure for range queries

- **Ongoing Work**
  - Complex queries
  - Load balancing
One Torus to Rule Them All: Multi-dimensional Queries in P2P Systems

(From WebDB 2004)

Prasanna Ganesan, Beverly Yang
Hector Garcia-Molina

Stanford University
Motivation

- **P2P Systems**
  - Dynamic set of nodes
  - Dynamic data distributed over nodes
  - No centralization

- **New P2P applications desire multi-dimensional queries**
  - Photo Sharing: Find all labels for photos in a geographical area
  - Multi-player games: Find all objects in an area
Problem

Devise P2P system to store relation R with:

1. Efficient tuple insertion/deletion
2. Efficient node join/leave
   - Minimize #messages
3. Efficient multi-dimensional range queries
   - Minimize #nodes processing query
4. Load balance across nodes
Challenge 1: Partitioning Problem

- Partition data with
  - Locality: Keep nearby tuples on same node
  - Load balance: Equal #tuples on all nodes

- Complications
  - Dynamic data
  - Dynamic nodes
Challenge 2: Routing Problem

- Route query/insert/delete to relevant nodes
  - No centralization!
  - Replicated directory too expensive!
  - Trade-off between cost of query and cost of maintaining routing structure
Roadmap

- Two Different Approaches
  - SCRAP: Space-filling curves with Range Partitions
  - MURK: Multi-dimensional Rectangulation with kd-trees
- Comparing the two approaches
SCARP Partitioning

Two-Step Process

1. Map data to 1-d with space-filling curve
   - E.g., \(<110011,010101>\) becomes \(101100011011\)
Scrap Partitioning (2)

2. Range partition 1-d data
   - Preserves locality!
Load Balancing with SCRAP

- Adjust partitions when unbalanced
  - Adjust boundary with neighbor
  - Migrate to new area
- Guarantees: All loads within factor 4.24. Constant tuple movements per insert/delete [GBGM04]
Query Routing

- Map multi-dim query to set of 1-d ranges
- Send each 1-d range query to relevant node
- Use a linked list to interconnect nodes
  - Add “skip” pointers for fast routing
- $O(\log n)$ messages for routing/node joins/leaves
Roadmap

- Two Different Approaches
  - SCRAP: Space-filling curves with Range Partitions
  - MURK: Multi-dimensional Rectangulation with kd-trees
- Comparing the two approaches
MURK

- Intuition: Partition data in native space into “Rectangles”
  - a la kd-trees
**Kd-tree Interpretation**

- Nodes form leaves of 
kd-tree
- Node Join: Split 
existing leaf
- Node leave
  - Sibling takes over
  - If no sibling, find 
someone in sibling sub-
tree
Murk Properties

- Locality:
  Rectangulation better than SCRAP

- Load Balance
  - Ok if data distribution is static
  - ??? If data distribution is dynamic
Routing Queries

- Build a grid of nodes
  - Adjacent nodes link to each other
  - Analogous to linked list in higher dimensions

- Problems
  - Node managing large space has many neighbors!
  - Routing on grid is too slow. Need skip pointers
  - Not easy to add skip pointers (see paper)
Evaluation

- **Datasets**
  - Uniform: 32-bit ints drawn at random
  - Skewed: Photo Co-ords from real collection

- **Nodes join one at a time to build network**

- **Evaluate**
  - Locality: #nodes that process a query
  - Routing: #messages transmitted per query
Dimensionality vs. Locality

$\#\text{nodes} = 8192.$

$\#\text{Ideal Locality} = 1$
Selectivity vs. Locality

![Graph showing selectivity vs. locality]
Network Size vs. routing Cost

![Graph showing network size vs. routing cost](image)
Conclusions

- **SCRAP**
  - Simple partitioning and routing
  - Excellent load balance
  - Issue: Space-filling curve offers poor locality

- **MURK**
  - Much better locality than SCRAP
  - Routing still ok
  - Load balance is more complex and heuristic 😞
More Information

- **Load Balancing, Range Queries and P2P**
  - “Online Balancing of Range-Partitioned Data with Applications to P2P Systems”, VLDB 2004
  - “Distributed Balanced Tables: Not Making a Hash of it All”, Stanford Tech Report
  - Google: “Prasanna Ganesan”

- **More work on P2P**
  - Google: “Stanford Peers”