Outline of the Talk

- Introduction – the Problem
- Dissemination of Information in Pub/Sub Digital Libraries
- Super-peer-based Digital Libraries
  - Hyper-graph Models for Point and Continuous Queries
- Hyper-graph Partitioning
- Hyper-graph Partitioning for Load Balancing
- The HyPer Partitioner
- Load Balancing via HyPer
  - Computational Results
- Conclusions
Introduction

The Problem:

- How to balance content metadata information in a Super-peer-based D.L. so that
  - Each super-peer contains a balanced amount of information in its database and correspondingly receives balanced amount of point and/or continuous queries regarding the documents in the D.L.
  - Queries (point or continuous) are based on attribute-value pairs
    - i.e. we ask for all documents whose specified attributes take on the specified on the query values
    - Documents returned for a query must satisfy all conditions of the query (conjunction).

- In a balanced system, average response time to such queries is minimized as each super-peer is responsible for approximately the same number of attributes and each attribute-value pair in a query is to be checked in one super-peer-node (or very few...)

- Storage requirements are also optimized in a balanced Super-peer-based D.L.
Distributed Pub/Sub Digital Library Model
Assume each document is described by a set of well-defined meta-data such as

- Authors’ names
- Year of publication
- Publisher, ISBN etc.
- Title
- Keywords
- etc.

Each profile (or continuous query) is described by the same set.
Hyper-Graph Model of Digital Content in D.L.

- HyperGraph Model of Documents/Profiles in a DL.
- Each net represents a document or profile.
- Nodes represent valued attributes
  - (e.g. author-last-name="Christou")
Hyper-Graph Partitioning for Load Balancing

- Objectives of Partitioning a Hyper-Graph
  - To partition the hyper-graph’s nodes among approximately equal-size node-partitions so that the cut hyper-edges are minimized.

- By partitioning e.g. the profiles hyper-graph into as many blocks as there are super-peers in the P2P network, we achieve load balancing in the continuous queries:
  - clean separation of the clusters resulting in much better performance from the system as a whole
    - few super-peer nodes involved in each query
    - much less data in each super-peer’s database.
Hyper-Graph Partitioning

- 3-phase algorithm

1. Keep coarsening the original Hyper-graph into successively coarser (and smaller in #nodes) hyper-graphs storing them in memory along the way.

2. When some criterion is satisfied (such as small enough size of coarse hgraph), apply move-based heuristic (e.g. F-M partitioning algorithm) to partition coarse hgraph.

3. Apply the partition to the previous level hgraph, refine the partition in the upper level using some local greedy heuristic, all the way up to the original hyper-graph.
HyPer Partitioner for Load Balancing in D.Ls

- HyPer is a multi-threaded multi-level hyper-graph partitioner with the following extra features:
  - Supports warm-starting
    - It is capable of starting with an existing partitioning solution and finding a local optimum solution in the neighborhood of the existing one.
    - Also much faster when warm-started
    - “neighborhood” defined with a number of criteria
  - Runs as a server process, listening to requests for partitioning
    - Spawns new thread to process new request.

- Refinement Engine: variant of the k-way Greedy heuristic of Karypis et. al. in khMetis)
- Coarsening Scheme: variant of the First Choice scheme (FC) of Karypis et. al. in khMetis).
  - Allows v- and V-cycles (may coarsen together only nodes belonging to the same prior partition)
Overall System Algorithm (running on each super-peer) for Continuous Queries

- Upon arrival of a new document:
  - parses all attribute-value pairs in the arrived item.
  - checks with a Distributed Hash-Table (DHT) to find super-peers responsible for the parsed attribute-value pairs;
    - (Bamboo-DHT)
  - if there is an attribute-value pair not encountered before it is assigned to the super-peer that is assigned to most of the other attribute-value pairs in the arrived item
    - the DHT is appropriately informed.
  - Each super-peer responsible for at least one of the attribute-value pairs of the arrived item is informed of the arrival and updates its database accordingly.
    - Uses its indexing structures and methods to find all matching profiles
    - notifies owners of those profiles about the new arrival.
- After a fixed number of arrivals, warm-start the partitioner to obtain a new solution to the modified hyper-graph
  - update the super-peer nodes’ databases accordingly.
- After a much larger fixed number of arrivals, do a complete cold-start of the hyper-graph partitioner to obtain a brand-new solution, closer to the global optimum.
Index Structures for Continuous Queries

- Variant of Count-Keys Method (Molina et. al.)
- DBMS implementation using MySQL
System Interactions
<table>
<thead>
<tr>
<th>2 Super-peers</th>
<th>HyPer</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>#profs</td>
<td>avg. resp. time</td>
<td>avg. resp. time</td>
</tr>
<tr>
<td>10K</td>
<td>0.15</td>
<td>0.17</td>
</tr>
<tr>
<td>100K</td>
<td>0.17</td>
<td>0.20</td>
</tr>
<tr>
<td>1M</td>
<td>0.36</td>
<td>0.95</td>
</tr>
</tbody>
</table>

Experiment conducted under a concurrent load of 200 requests
# Average Response Times for Network of 4 Super-Peers

<table>
<thead>
<tr>
<th>#profs</th>
<th>HyPer</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>10K</td>
<td>0.07</td>
<td>0.09</td>
</tr>
<tr>
<td>100K</td>
<td>0.09</td>
<td>0.12</td>
</tr>
<tr>
<td>1M</td>
<td>0.29</td>
<td>0.79</td>
</tr>
</tbody>
</table>

Experiment conducted under a concurrent load of 200 requests.
## Average Response Times for a Network of 8 Super-Peers

<table>
<thead>
<tr>
<th>#profs</th>
<th>avg. resp. time</th>
<th>avg. resp. time</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYPER</td>
<td>0.04</td>
<td>0.04</td>
</tr>
<tr>
<td>10K</td>
<td>0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>100K</td>
<td>0.05</td>
<td>0.07</td>
</tr>
<tr>
<td>1M</td>
<td>0.22</td>
<td>0.47</td>
</tr>
</tbody>
</table>

Experiment conducted under a concurrent load of 200 requests
Conclusions – Future Directions

- The experimental results for a synthetic library of publications
  - (#attribute-value pairs = 100,000)
  - distribution among the documents follows Zipf’s law.
  - 4,000,000 profiles
  - 8-node Super-peer P2P network
- The average response time in the presence of 200 concurrent publication requests was approximately 0.61 seconds. The storage overheads were 4.5%, and the average number of matching profiles for each incoming publication was 1700.
- Thus, running times as well as storage overheads seem to be in an reasonable frame.
- For the future:
  - Handle the problem of gradually “shifting away from an optimal partition” to the extent that cold-start repartitioning results in significant need for (meta-)data transfers.
  - Study the trade-offs of the approach under such circumstances.
Load Balancing Issues In Super-Peer-based Pub/Sub Digital Libraries

Thank you!