Database Support for Recommender Systems

Mohamed F. Mokbel

Department of Computer Science and Engineering, University of Minnesota
www.cs.umn.edu/~mokbel
mokbel@cs.umn.edu
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- RecBench: A Benchmark for Recommender System Architectures
- RecStore: A Storage Engine Support for Recommender Systems
- LARS: A Location-Aware Recommender System
- Recathon: A Context-Aware Recommender System
- Summary, Commercial Ads, and Acknowledgments
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- **RecBench**: A *Benchmark for Recommender System Architectures*
- **RecStore**: A *Storage Engine Support for Recommender Systems*
- **LARS**: A *Location-Aware Recommender System*
- **Recathon**: A *Context-Aware Recommender System*
- Summary, Commercial Ads, and Acknowledgments
Recommender Systems

- Analyze user behavior to recommend personalized and interesting things to do/read/see

- **Collaborative filtering** process is the most commonly used one in Recommender Systems
Collaborative Filtering (CF)

<table>
<thead>
<tr>
<th></th>
<th>Movie 1</th>
<th>Movie 2</th>
<th>Movie 3</th>
<th>Movie 4</th>
<th>Movie 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★</td>
<td>★★</td>
<td>★★★★</td>
</tr>
<tr>
<td>2</td>
<td>★★★</td>
<td>★★★</td>
<td>★★★★</td>
<td>★★★</td>
<td>?</td>
</tr>
<tr>
<td>3</td>
<td>?</td>
<td>★★</td>
<td>★★★★</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>4</td>
<td>★★★★</td>
<td>★★</td>
<td>★★★</td>
<td>★★</td>
<td>★★★</td>
</tr>
<tr>
<td>5</td>
<td>★★★★</td>
<td>★★★★</td>
<td>?</td>
<td>★★★</td>
<td>★★★</td>
</tr>
</tbody>
</table>

February 2013
### Item-Based CF Model Building

#### Similarity measures

1. Cosine distance
2. Pearson correlation
3. Spearman correlation
4. Adjusted cosine distance

#### Similarity ($m_1, m_3$)

<table>
<thead>
<tr>
<th></th>
<th>$m_1$</th>
<th>$m_2$</th>
<th>$m_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_1$</td>
<td>4.0</td>
<td>3.0</td>
<td>3.5</td>
</tr>
<tr>
<td>$m_2$</td>
<td>3.0</td>
<td>4.0</td>
<td>3.0</td>
</tr>
<tr>
<td>$m_3$</td>
<td>3.5</td>
<td>3.0</td>
<td>4.0</td>
</tr>
</tbody>
</table>

...
**Item-Based CF Recommendations**

<table>
<thead>
<tr>
<th>Item</th>
<th>Similarity</th>
<th>Rating 1</th>
<th>Rating 2</th>
<th>Rating 3</th>
<th>Rating 4</th>
<th>Rating 5</th>
<th>Rating 6</th>
<th>Rating 7</th>
<th>Rating 8</th>
<th>Rating 9</th>
<th>Rating 10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MANHATTAN</strong></td>
<td><strong>2 stars</strong></td>
<td>0.8</td>
<td>0.9</td>
<td>0.7</td>
<td>0.9</td>
<td>0.7</td>
<td>0.9</td>
<td>0.7</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
</tr>
<tr>
<td><strong>MANHATTAN</strong></td>
<td><strong>3.5 stars</strong></td>
<td>0.6</td>
<td>0.8</td>
<td>0.6</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td><strong>MANHATTAN</strong></td>
<td><strong>4 stars</strong></td>
<td>0.4</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
</tbody>
</table>

For example:

- $(0.8 \times 2 + 0.6 \times 4) / (0.8 + 0.6) = 2.86$
- $(0.9 \times 2 + 0.5 \times 4) / (0.9 + 0.5) = 2.71$
- $(0.5 \times 2 + 0.7 \times 4) / (0.7 + 0.5) = 3.17$
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- **RecBench**: A Benchmark for Recommender System Architectures
- **RecStore**: A Storage Engine Support for Recommender Systems
- **LARS**: A Location-Aware Recommender System
- **Recathon**: A Context-Aware Recommender System
- Summary, Commercial Ads, and Acknowledgments

Recommender Systems: Quality vs. Performance

- Recommender systems community have only focused on “quality” issues; “performance” is considered a secondary issue.

“We have chosen not to discuss computation performance of recommender algorithms. Such performance is certainly important, and in the future we expect there to be work on the quality of time-limited and memory-limited recommendations.”


“[Our] solution is based on a huge amount of models and predictors which would not be practical as part of a commercial recommender system. However, this result is a direct consequence of the nature and goal of the competition: obtain the highest possible accuracy at any cost, disregarding completely the complexity of the solution and the execution performance.”

Team BelKor’s Pragmatic Chaos, Winner of the 2009 Netflix Prize

- All heavy work are done offline
- Models are built over long time period, e.g., movie or books ratings
- The rank of one item in the system slowly change
Things have changed...

- We live in an increasingly social and “real-time” world
  - Number of things to recommend is growing exponentially
  - Users expressing opinions faster than ever
  - Recommendations change second-to-second

“Offline” step can no longer be tolerated
Recommender Systems in DBMS?

- Incoming stream of ratings data: \((\text{user}, \text{item}, \text{rating})\)

- Ratings are used to build a recommendation model as:
  - Item-based collaborative filtering: \((\text{item}, \text{item}, \text{similarity})\)
  - User-based collaborative filtering: \((\text{user}, \text{user}, \text{similarity})\)

- Recommendation query:
  - Item-based collaborative filtering:
    - Given a user \(u\), find the top-\(k\) items that are most similar to the items that \(u\) has liked before
  - User-based collaborative filtering:
    - Given a user \(u\), find the top-\(k\) items that the users who are similar to \(u\) have liked

Recommender Systems have all the ingredients of a data management problem
DBMS Challenge

Lets not try to find a new way of doing recommendation*

* ACM RecSys community is already doing excellent job in this frontier. Lets start from there.

We need to provide online support and scale up the computations of existing recommender methods.

Can DBMS do it?
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- **RecBench:** *A Benchmark for Recommender System Architectures*
- **RecStore:** *A Storage Engine Support for Recommender Systems*
- **LARS:** *A Location-Aware Recommender System*
- **Recathon:** *A Context-Aware Recommender System*
- Summary, Commercial Ads, and Acknowledgments
**MovieLens**

Welcome to MovieLens!
Free, personalized, non-commercial, ad-free, great movie recommendations.

Have questions? Take the MovieLens Tour for answers.

Not a member? Join MovieLens now.

Need a gift idea? Try MovieLens QuickPick!

New to MovieLens?
Join today!
You get great recommendations for movies while helping us do research. Learn more:
- Try out QuickPick: Our Movie Gift Recommender
- Take the MovieLens Tour
- Read our Privacy Policy
- See our Browser Requirements
- Learn about Our Research

Hello MovieLens Users!
Please login to be directed to your proper location
Please log in:
Username/E-mail: [Input]
Password: [Input]
Save login: [Checkbox]
Log into MovieLens
Forgot your password?
New member? Join now

MovieLens is a free service provided by GroupLens Research at the University of Minnesota. We sometimes study how our members use MovieLens in order to learn how to build better recommendation systems. We promise to never give your personal information to anyone; see our privacy policy for more information.

**ACM Software Award 2010:** – GroupLens Collaborative Filtering Recommender Systems:
Peter Bergstrom, Lee R Gordon, Jonathan L Herlocker, Neophytos Iacovou, Joseph A Konstan, Shyong (Tony) K. Lam, David Maltz, Sean McNee, Bradley N Miller, Paul J Resnick, John T Riedl, Mitesh Suchak

**MovieLens:** A Movie Recommender System, built and maintained at University of Minnesota (GroupLens Research)
- 10 Million ratings
- 10,000 Movies
- 72,000 Users
RecBench:
A Benchmark for Recommender System Architectures

Goals:
① Prompt DB & RecSys research communities to work together
② A benchmark to test performance of different system architectures

Six common recommendation tasks are carefully selected

Every task is implemented on three different architectures

MultiLens
- “Hand-built” system
- Code optimized for item-based CF
- Uses DBMS for metadata and text-search queries

PostgreSQL
- Unmodified DBMS
- Ratings relation: ratings(usr, itm, rating)
- Model relation: model(itm, itm, sim)
- All tasks implemented in standard SQL

Custom DBMS
- DBMS (PostgreSQL) modified to optimize for fast recommender model updates
- SQL same as unmodified DBMS approach
RecBench – Task 1: Initialization

Prepare system to start serving user recommendations

Stored Recommendation Model
RecBench – Task 2: Pure Recommend

Produce top-k recommendations from system’s entire item pool.

![Image of MovieLens interface showing top picks and recommendations.]

February 2013
RecBench – Task 3: Filtered Recommend

Produce top-k recommendations that match item constraints
Produce top-k recommendations based on blended text-search and recommendation score
RecBench – Task 5: Item Prediction

Generate a user’s predicted rating for a target item
Incorporate new item(s) into the system for recommendation

Incorporate new item(s) into the system for recommendation

Submit a Movie Title (help)

IMDb URL

Find the movie you want to add on IMDb and enter its URL. We use this to check if the movie is already in MovieLens.
RecBench: Summary of Results

- **Datasets**
  - MovieLens: 10 M movie ratings, 10K movies, and 72K users
  - Netflix Challenge: 100 M movie ratings, 18K movies, and 480K users

- **Tasks 1 & 2 (Initialization & Pure Recommend)**
  - PostgreSQL has by far the worst performance
  - Custom DBMS does a good job but not as excellent as MultiLens

- **Tasks 3 & 4 (Filtered & Blended Recommend)**
  - CustomDBMS way outperforms MultiLens as it takes advantage of its select & top-k operators
  - PostgreSQL performance in the middle.

- **Tasks 5 & 6 (Item Prediction & New Items)**
  - MultiLens outperforms others in Task 5 (a basic component in Task 1)
  - CustomDBMS outperforms others in Task 6 due to the built-in incrementally maintained statistics
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- **RecBench**: A Benchmark for Recommender System Architectures
- **RecStore**: A Storage Engine Support for Recommender Systems
- **LARS**: A Context-Aware Recommender System
- **Recathon**: A Context-Aware Recommender System
- Summary, Commercial Ads, and Acknowledgments

RecStore: A Storage Engine Support for Recommender Systems

RecStore pushes the recommender model building inside the Database Engine to provide online support and scale up the computations of existing recommender methods.

- **Adaptivity of RecStore**
  - RecStore is adaptive to different system workloads (Query Intensive vs. Update Intensive)

- **Extensibility of RecStore**
  - RecStore is extensible to support many recommendation methods (e.g., item-based CF, user-based CF).
RecStore: Architecture

Access Methods (Index, Scan)

Recommendation Queries

Rating Updates

Rating Data

Intermediate Store

Model Table

Intermediate Filter

Model Filter

RecStore
RecStore: Query Latency vs. Maintenance Cost

- **Materialize-All**
  - Low Latency Recommendation Query.
  - High Storage and maintenance Cost.

- **Materialize-None**
  - High Latency Recommendation Query
  - Low Storage and maintenance Cost.

- **Intermediate Store Only**
  - Middle Ground between Materialize-All and Materialize-None

- **Intermediate Store / Partial Model Store**
  - Middle Ground between Materialize-All and Intermediate-Only

- **Partial Intermediate Store / Partial Model Store**
  - Lies between Partial Model and Intermediate Only
DEFINE RECSTORE MODEL ItemItemCosine
FROM Ratings R1, Ratings R2
WHERE R1.ItemId <> R2.itemId AND R1.userId = R2.userId

WITH INTERMEDIATE STORE:
(R1.itemId as item, R2.itemId as rel_itm,
 vector_lenp, vector_lenq, dot_prod, co_rate)

WITH INTERMEDIATE FILTER:
ALLOW UPDATE WITH My_IntFilterLogic(),
UPDATE vector_lenp AS vector_lenp + R1.rating * R1.rating,
UPDATE vector_lenq AS vector_lenq + R2.rating * R2.rating,
UPDATE dot_prod AS dot_prod + R1.rating * R2.rating,
UPDATE co_rate AS 1

WITH MODEL STORE:
(R1.itemId as item, R2.itemId as rel_itm, COMPUTED sim)

WITH MODEL FILTER:
ALLOW UPDATE WITH My_ModFilterLogic(),
UPDATE sim AS if (co_rate < 50)
    co_rate * dot_prod / (50*sqrt(vector_lenp) * sqrt(vector_lenq));
else
    co_rate / sqrt(vector_lenp) * sqrt(vector_lenq);
RecStore: Querying a Recommender Model

User/item ratings

<table>
<thead>
<tr>
<th>uid</th>
<th>mid</th>
<th>rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3.5</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

/* Find movies rated by REC_USER_X, * store in temp table usrXMovies */
CREATE TEMP TABLE usrXMovies AS
SELECT R.mid as itemId, R.rating as rating
FROM ratings R
WHERE R.uid = REC_USER_X;

/* Generate predictions using weighted sum */
SELECT M.itm as Candidate Item,
SUM(M.sim * U.rating)/ SUM(M.sim) as Prediction
FROM Model M, usrXMovies U
WHERE M.rel_itm = U.itmId AND
M.itm NOT IN (select itmId FROM usrXMovies)
GROUP BY M.itm ORDER BY Prediction DESC;

Maintain the recommendation Model to efficiently answer recommendation queries
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- RecBench: A Benchmark for Recommender System Architectures
- RecStore: A Storage Engine Support for Recommender Systems
- LARS: A Location-Aware Recommender System
- Recathon: A Context-Aware Recommender System
- Summary, Commercial Ads, and Acknowledgments

Mohamed Sarwat, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. "LARS*: A Scalable and Efficient Location-Aware Recommender System". IEEE Transactions on Knowledge and Data Engineering, TKDE 2013, To Appear.

“Locations” and “Recommendations”

- Recommender systems rely on the input triple: 
  \[(user, item, rating)\]
  - Recommender systems completely ignore the spatial aspects of both users and items

Do we need to consider “Locations”? 

- Adding Location-awareness to Recommender Systems
  - Recommend movies based on the locations of the ratings

- Adding Recommendation-awareness to Location-based services
  - Instead of asking about restaurants in a certain area or closest to me, I can ask a recommender system to suggest few restaurants to me
Location Matters: Netflix Rental Patterns

- Movie preferences differ based on the user location (zip code)

Most rented in 55418
1. Milk
2. The Curious Case of Benjamin Button
3. Burn After Reading
4. The Wrestler
5. Slumdog Millionaire
6. Gran Torino
7. Doubt
8. Changeling
9. Rachel Getting Married
10. Twilight
...
16. I Love You, Man

Most rented in 55113
1. The Curious Case of Benjamin Button
2. Slumdog Millionaire
3. Gran Torino
4. Doubt
5. Milk
6. Seven Pounds
7. Burn After Reading
8. Changeling
9. The Wrestler
10. New in Town
...
24. I Love You, Man

Most rented in 55455
1. I Love You, Man
2. Slumdog Millionaire
3. Adventureland
4. My Best Friend's Girl
5. Nick and Norah's Infinite Playlist
6. Sunshine Cleaning
7. Forgetting Sarah Marshall
8. Away We Go
9. Role Models
10. Confessions of a Shopaholic
11. precinct ebooks

Most rented in 55404
1. Burn After Reading
2. Milk
3. The Curious Case of Benjamin Button
4. Slumdog Millionaire
5. The Wrestler
6. Twilight
7. Doubt
8. Rachel Getting Married
9. Changeling
10. Gran Torino
...
12. I Love You, Man
Location Matters: Top-3 Check-In Destinations in Foursquare

Destination preferences differ based on the user location (zip code) and the destination location.

Foursquare users from Robbinsdale tend to visit venues in ...

<table>
<thead>
<tr>
<th>City</th>
<th>% of check-ins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brooklyn Park</td>
<td>32%</td>
</tr>
<tr>
<td>Robbinsdale</td>
<td>20%</td>
</tr>
<tr>
<td>Minneapolis</td>
<td>15%</td>
</tr>
</tbody>
</table>

Foursquare users from Falcon Heights tend to visit venues in ...

<table>
<thead>
<tr>
<th>City</th>
<th>% of check-ins</th>
</tr>
</thead>
<tbody>
<tr>
<td>St. Paul</td>
<td>17%</td>
</tr>
<tr>
<td>Minneapolis</td>
<td>13%</td>
</tr>
<tr>
<td>Roseville</td>
<td>10%</td>
</tr>
</tbody>
</table>

Foursquare users from Edina tend to visit venues in ...

<table>
<thead>
<tr>
<th>City</th>
<th>% of check-ins</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edina</td>
<td>59%</td>
</tr>
<tr>
<td>Minneapolis</td>
<td>37%</td>
</tr>
<tr>
<td>Edin Prarie</td>
<td>5%</td>
</tr>
</tbody>
</table>
We need to go beyond the traditional rating triple \((user, item, rating)\) to include the following taxonomy:

1. **Spatial Rating for Non-spatial Items**
   - \((user\_location, user, item, rating)\)
   - **Example**: A user with a certain location is rating a movie
   - **Recommendation**: Recommend a movie that neighbor users have liked

2. **Non-spatial Rating for Spatial Items**
   - \((user, item\_location, item, rating)\)
   - **Example**: A user with unknown location is rating a restaurant
   - **Recommendation**: Recommend a restaurant within a close vicinity

3. **Spatial Rating for Spatial Items**
   - \((user\_location, location, item\_location, item, rating)\)
   - **Example**: A user with a certain location is rating a restaurant
Spatial User Ratings For Non-Spatial Items

1. Partition ratings by user location

Cell 1

Cell 2

Cell 3

2. Build collaborative filtering model for each cell using only ratings contained within the cell

Cell 1

Build Collaborative Filtering Model using:

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

Cell 2

Build Collaborative Filtering Model using:

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>4</td>
</tr>
</tbody>
</table>

Cell 3

Build Collaborative Filtering Model using:

<table>
<thead>
<tr>
<th>User</th>
<th>Item</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

3. Generate recommendations using collaborative filtering using the model of the cell containing querying user

Recommendation List

Cell 1

Cell 2

Cell 3

Querying user
Spatial User Ratings For Non-Spatial Items

- Smaller cells → More “localized” answer
- Each user can select a personalized localized level
- Scalability problem in terms of maintaining large numbers of recommendation models

- No need to maintain all cells
- If four cells will end up giving the same recommendations, merge them.
- If ratings inside a cell are diverse, split it
- Merging and splitting balance between localization and storage/maintenance
Non-Spatial User Ratings For Spatial Items

- Penalize items based on their distance from the user.
- Distance from the user is normalized to the ratings scale to get the Travel Penalty.
- Use a ranking function that combines the recommendation score and travel penalty.
- Incrementally, retrieve items based on travel penalty, and calculate the ranking score on an ad-hoc basis.
- Employ an early stopping condition to minimize the list of accessed items to get the K recommended items.
Spatial User Ratings For Spatial Items

User Partitioning + Travel Penalty

Travel Penalty
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- **RecBench**: A Benchmark for Recommender System Architectures
- **RecStore**: A Storage Engine Support for Recommender Systems
- **LARS**: A Location-Aware Recommender System
- **Recathon**: A Context-Aware Recommender System
- Summary, Commercial Ads, and Acknowledgments

Context-Aware Recommendation: Why?

A need for a system that generates context-aware recommendations.

“Recommend me five books”

“Recommend me five books, based on my location”

“Recommend me five books, based on my age”

“Recommend me five books, based on my job”

“Recommend me five books, based on my budget”
Recathon:
A Context-Aware Recommender System

Main Idea: Treat recommender systems in the same way as indexing in databases

- **Same as Indexing:**
  - A recommender can be built on one (or more) attribute(s)
  - A recommender can be dropped anytime
  - A recommender is maintained with inserting new items
  - There are different methods of building a recommender

- **Different from Indexing**
  - A query needs to explicitly specify which recommender model to use
  - Recommenders are maintained differently based on query and transaction workload
  - Recommenders can be maintained partially to provide part of the final answer or fully to directly give the final answer
Recathon: Creating a Recommender

CREATE RECOMMENDER
USERS FROM User_Table_Name
ITEMS FROM Items_Table_Name
RATINGS FROM Rating_Table_Name
ATTRIBUTES User_Attributes
USING RecommenderMethod

<table>
<thead>
<tr>
<th>UserID</th>
<th>Age</th>
<th>City</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>Minneapolis</td>
<td>3K</td>
</tr>
<tr>
<td>2</td>
<td>35</td>
<td>Saint Paul</td>
<td>4K</td>
</tr>
<tr>
<td>3</td>
<td>18</td>
<td>Falcon Heights</td>
<td>3.5K</td>
</tr>
<tr>
<td>4</td>
<td>23</td>
<td>Edina</td>
<td>5K</td>
</tr>
<tr>
<td>5</td>
<td>31</td>
<td>Minnetonka</td>
<td>10K</td>
</tr>
</tbody>
</table>

CREATE RECOMMENDER AgeRec
USERS FROM MovieUsers
ITEMS FROM MovieTable
RATINGS FROM MovieRating
ATTRIBUTES Age
USING ItemBasedCF

CREATE RECOMMENDER AgeCityRec
USERS FROM MovieUsers
ITEMS FROM MovieTable
RATINGS FROM MovieRating
ATTRIBUTES Age, City
USING SVD

<table>
<thead>
<tr>
<th>ItemID</th>
<th>Movie</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lord of the Rings</td>
</tr>
<tr>
<td>2</td>
<td>Manhattan</td>
</tr>
<tr>
<td>3</td>
<td>The Good, the Bad, and the Ugly</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UserID</th>
<th>ItemID</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>3.5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4.5</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1.5</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>5.0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3.0</td>
</tr>
</tbody>
</table>
Recathon: Querying a Recommender

- Once a recommender is created, a set of intermediate tables and views are created
- Tables and views are continuously maintained, based on an adaptive maintenance technique
- Recommenders are exposed to Recathon users as views that can be queried with standard SQL

```
SELECT ItemID
FROM AgeRec R1
RECOMMEND(10)
R1.uid = 1 AND
R1.age = 20
```

Recommend me a movie based on my Age

```
SELECT ItemID
FROM AgeRec R1
RECOMMEND(10)
R1.uid = 1 AND
R1.age = 20 AND
City = 'Edina'
```

Recommend me a movie based on my Age & City
Talk Outline

- Background on Recommender Systems
- DBMS for Recommender Systems
- RecBench: *A Benchmark for Recommender System Architectures*
- RecStore: *A Storage Engine Support for Recommender Systems*
- LARS: *A Context-Aware Recommender System*
- Recathon: *A Context-Aware Recommender System*
- Summary, Commercial Ads, and Acknowledgments
Summary

Rating Updates

“Recommend me five books”

“Recommend me five books, based on my location”

“Recommend me five books, based on my age”

“Recommend me five books, based on my job”

“Recommend me five books, based on my budget”
Related Publications

Papers

Mohamed Sarwat, Justin J. Levandoski, Ahmed Eldawy, and Mohamed F. Mokbel. "LARS*: A Scalable and Efficient Location-Aware Recommender System". IEEE Transactions on Knowledge and Data Engineering, TKDE 2013, To Appear.


Demos


Commercial Ads
Efficient Spatial Operations

Analyze your data on large clusters with built-in spatial operations that run efficiently using spatial indexes

Website: http://spatialhadoop.cs.umn.edu/
Download source code, binary distribution, and instructions
MNTG: Web-based Traffic Generator

- Easy-to-generate traffic data for road networks
  - No need to do the installation/configuration
  - Very easy to get the data, just clicks
  - Works for road networks in US
  - Dedicated server for data generation
  - Email notifications
  - Visualization tools

Website: [http://mntg.cs.umn.edu](http://mntg.cs.umn.edu)
Video: [http://www.youtube.com/watch?v=dVP4oc0k9nU](http://www.youtube.com/watch?v=dVP4oc0k9nU)
Acknowledgment
The RecBench, RecStore, LARS, and Recathon Team

- **Ph.D. Alumni**

  Justin Levandoski  
  *(PhD, 2011)*  
  Researcher at Microsoft Research (MSR) -- Database Group,  
  Redmond, USA

- **Current Members: Ph.D. Students**

  James Avery  
  Ahmed Eldawy  
  Mohamed Sarwat
Other Group Members

■ Ph.D. Alumni

Chi-Yin Chow
(PhD, 2010)
Assistant Professor at City University of Hong Kong, Hong Kong

Biplob Debnath
(PhD, 2011)
Researcher, NEC Labs, NJ, USA

Mohamed Khalefa
(PhD, 2011)
Assistant Professor, Alexandria University, Egypt

■ Current Members: Ph.D. Students

Louai Al-Arabi
Jie Bao
Abdeltawab Hendawi
Amr Magdy
Acknowledgment: Funding

- **NSF-IIS:** Towards Spatial Database Management Systems for Flash Memory Storage. 2012 -2015

- **Microsoft Research.** *Microsoft Unrestricted Gift, October, 2010*

- **NSF- CAREER:** Extensible Personalization of Spatial and Spatio-temporal Database Management Systems. 2010 -2015

- **Microsoft Research.** *Microsoft Unrestricted Gift, January, 2010*

- **Microsoft Research.** *Microsoft Unrestricted Gift, April, 2009*

- **NSF- IIS:** Towards Ubiquitous Location Services: Scalability and Privacy of Location-based Continuous Queries. 2008 -2012

- **NSF- IIS:** Preference- And Context-Aware Query Processing for Location-based Data-based servers. 2008 -2012

Thanks