A *Customizable* Framework for Recommending OLAP Queries

A. Giacometti, P. Marcel, E. Negre
LI, Université François Rabelais Tours, France

Elsa.Negre@univ-tours.fr

CIKM / DOLAP'08

Outline

1) What is the problem?

Motivations, Intuitions and Example

2) How to solve the problem?

Our framework

3) Experimentation

4) Conclusion and Future work

Motivations and Intuitions (1)

Navigate an OLAP cube

- an analysis session
- the forthcoming query?

How to propose to the user his forthcoming query ?

Motivations and Intuitions (2)

Existing methods in:

Information Retrieval

Web Usage Mining

↓

Exploitation of the other users former navigations to generate recommendations

Example (1)

- An OLAP server used by several analysts

- Other users former analysis sessions:

$$\left.\begin{array}{l} S_1 = <q_1, q_2, q_3, q_4> \\ S_2 = <q_5, q_6, q_7> \\ S_3 = <q_8, q_9, q_{10}> \end{array}\right\} \text{Logged}$$

- A new session:
  - The current session:

$$S_c = <q^c_1, q^c_2>$$

$$\boxed{q^c_3\ ?}$$

Example (2)

- Problem 1: Sparsity of the log

➡ Generalization: query → class

Generalized sessions

$S_1 = <q_1, q_2, q_3, q_4>$

$S_2 = <q_5, q_6, q_7>$

$S_3 = <q_8, q_9, q_{10}>$

⬇

$g_1 = <c_1, c_2, c_3, c_4>$

$g_2 = <c_2, c_3, c_5>$

$g_3 = <c_4, c_3, c_5>$

$S_c = <q^c_1, q^c_2>$

⬇

$g_c = <c_2, c_3>$

Example (3)

- Problem 2: How to compute candidate recommendations ?

1) Matching generalized sessions and g_c:

g_c = subsequence of g_1 and g_2

$g_1 = \langle c_1, \boxed{c_2, c_3}, c_4 \rangle$ $g_c = \langle c_2, c_3 \rangle$

$g_2 = \langle \boxed{c_2, c_3}, c_5 \rangle$

2) Obtaining candidate classes: the successors

$\{c_4, c_5\}$

3) Obtaining the query representing a class:

$\{q_4, q_7\}$

Example (4)

- Problem 3: Ranking the candidate queries

➡ a ranking criterion

For example :

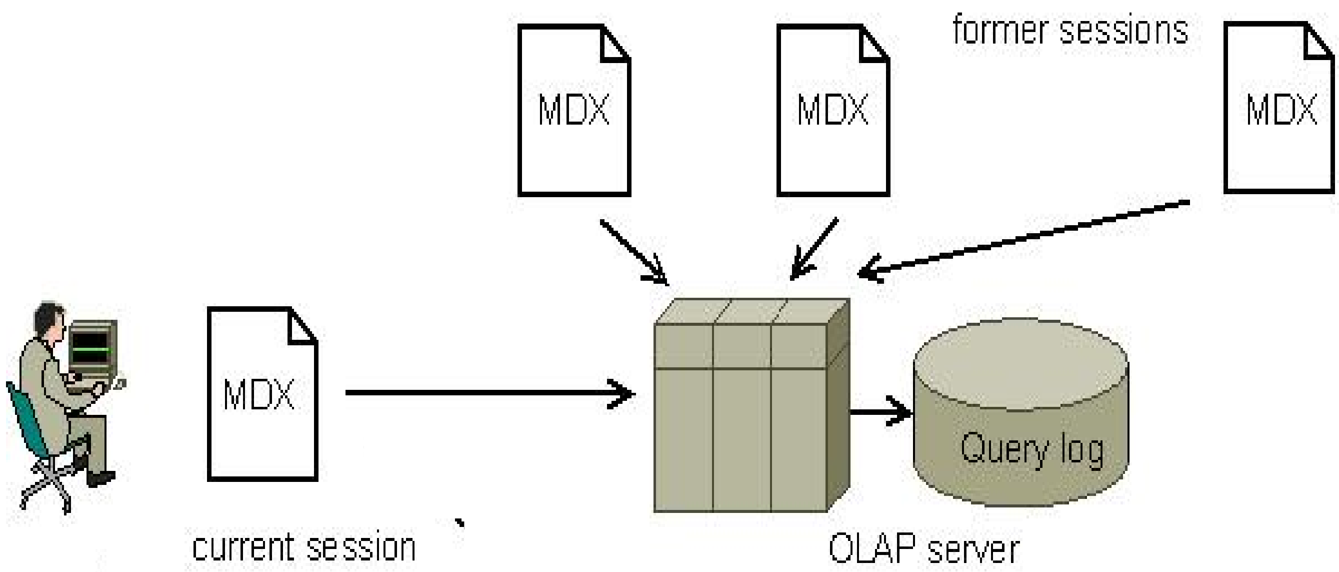
closeness to the last query of the current session

Recommendation = q_7

And then q_4 if the user is not happy with q_7…

Our *Customizable* Framework (1)

6 parameterized steps



Our *Customizable* Framework (2)

6 parameterized steps

- Partitioning the log

Our *Customizable* Framework (3)

- **A query set partitioning:**

  k-medoids algorithm

  - **a distance between queries:**

    Hausdorff distance for MDX queries

  ⇨ *A set of classes of queries*

- Example :

  $c_1=\{q_1\}$, $c_2=\{q_2, q_5\}$,

  $c_3=\{q_3, q_6, q_9\}$, $c_4=\{q_4, q_8\}$,

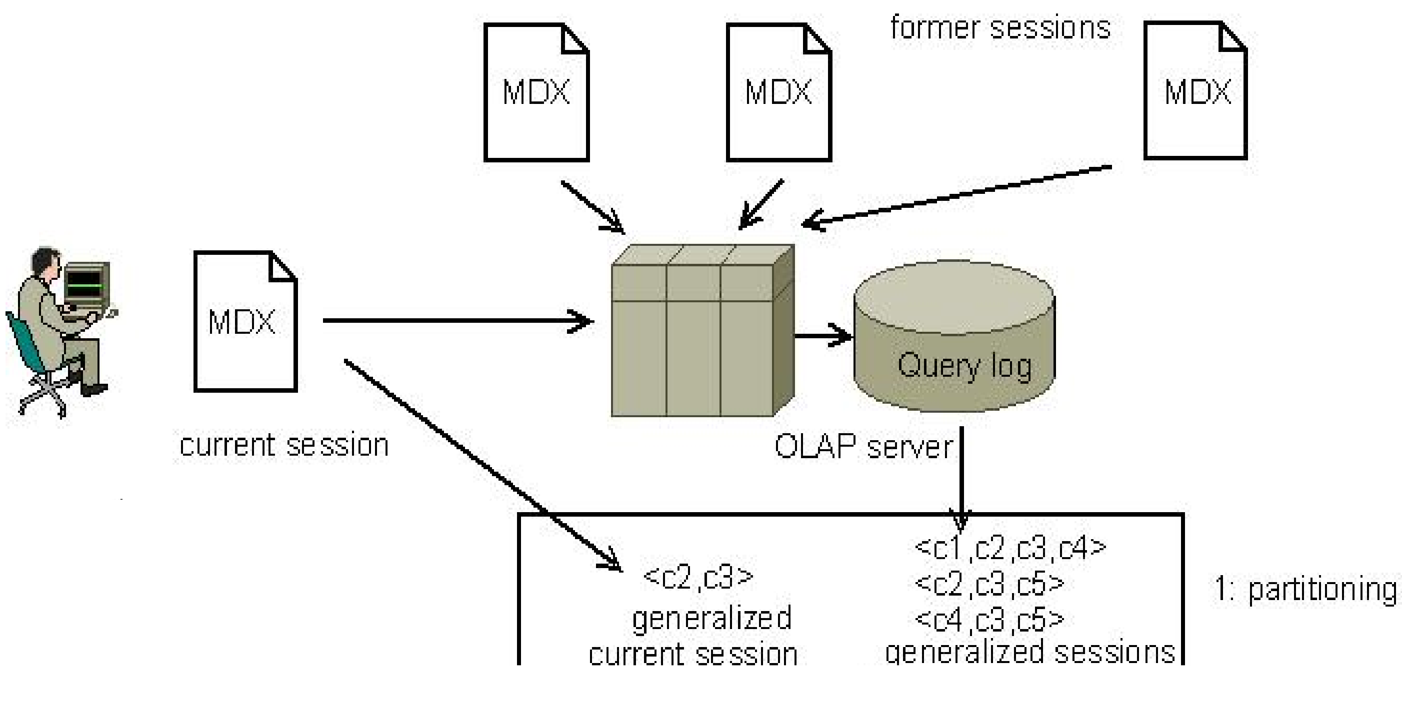
  $c_5=\{q_7, q_{10}\}$

Our *Customizable* Framework (4)

6 parameterized steps

- Partitioning the log
- Generalizing the sessions

Our *Customizable* Framework (5)



Our *Customizable* Framework (6)

6 parameterized steps

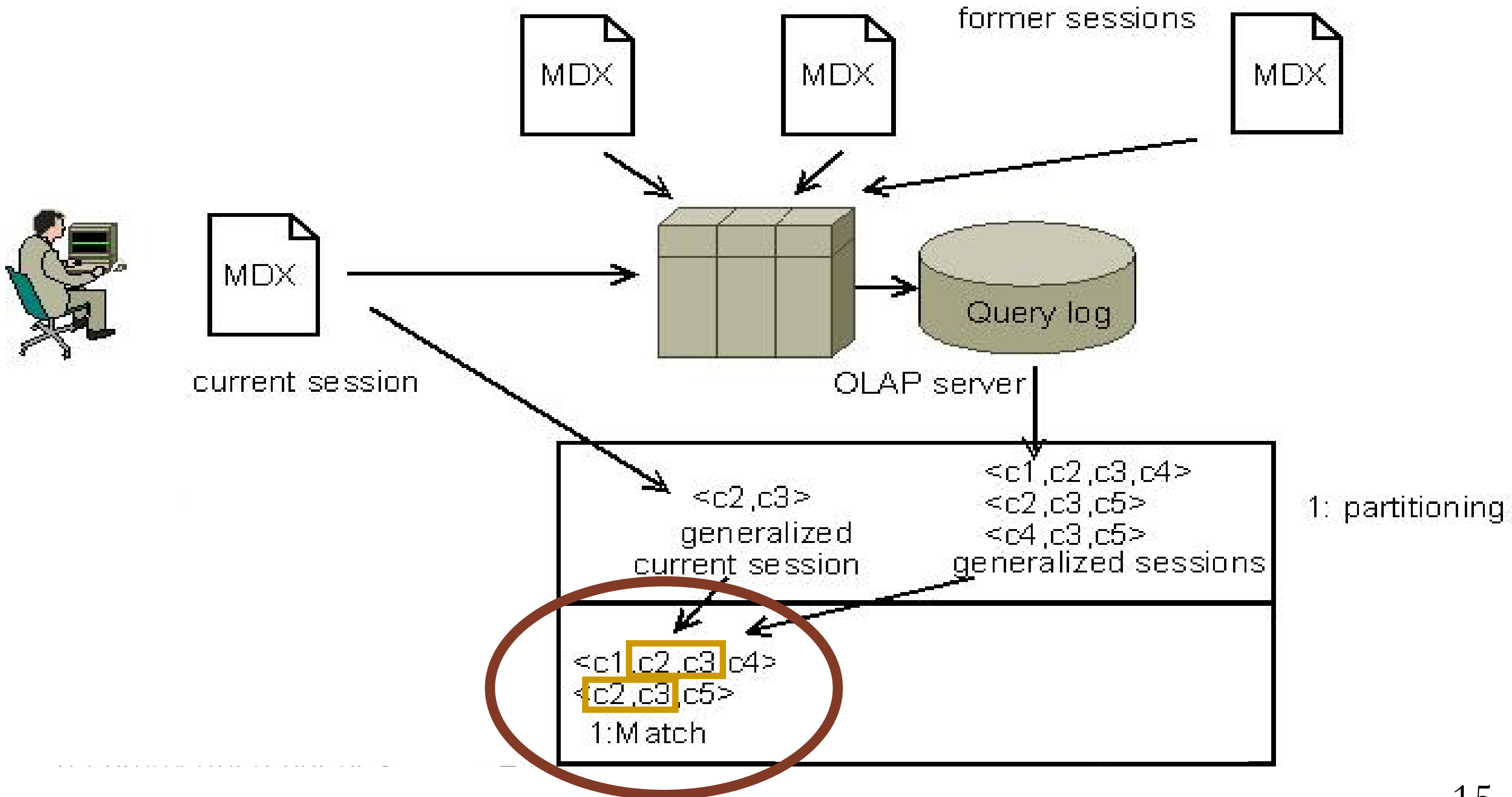
- Partitioning the log
- Generalizing the sessions
- Matching generalized sessions and the generalized current session

Our *Customizable* Framework (7)

- **Matching function:**

Approximate String Matching approach

→ *A set of candidate generalized sessions*

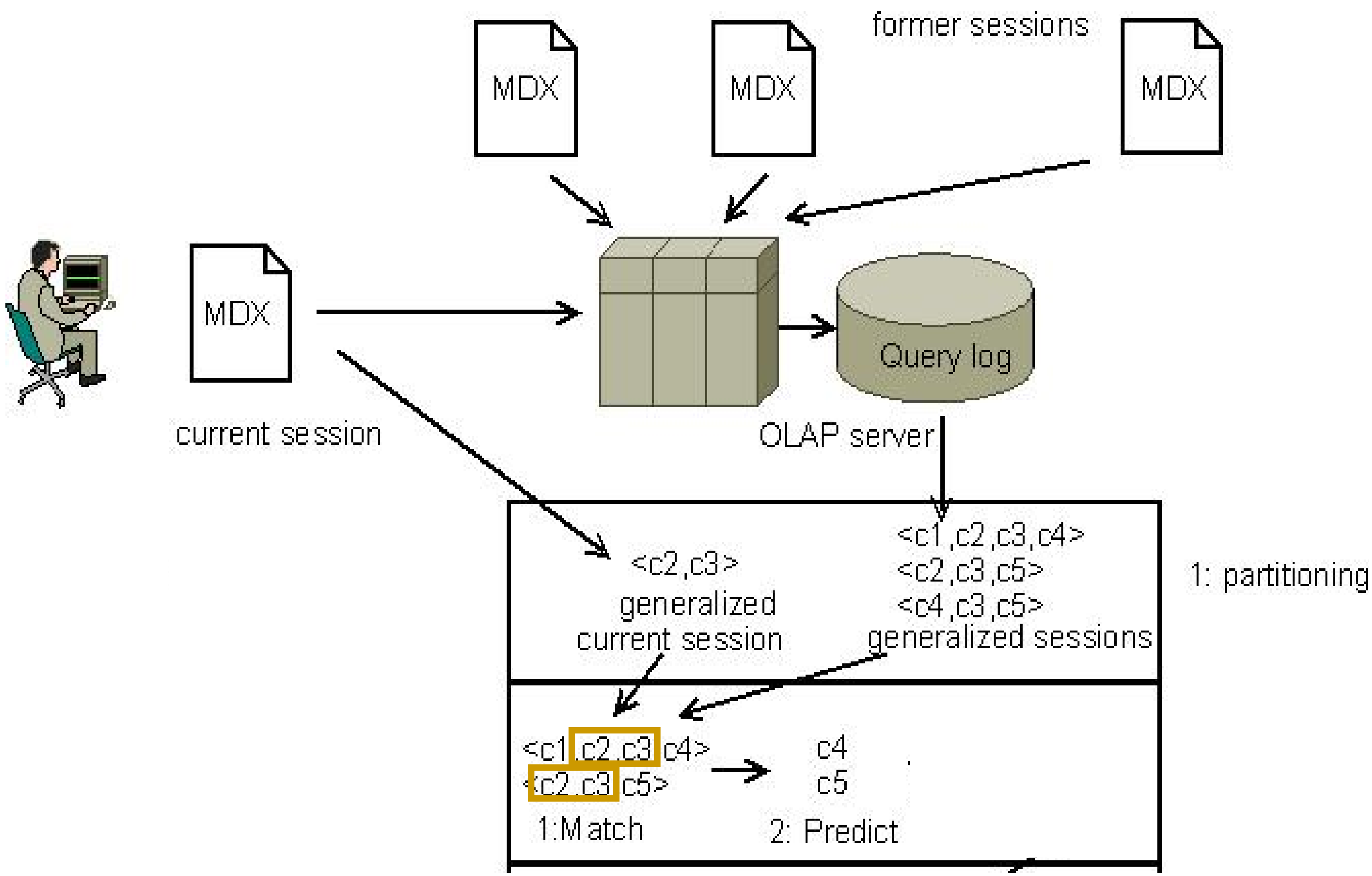


Our *Customizable* Framework (8)

6 parameterized steps

- Partitioning the log
- Generalizing the sessions
- Matching generalized sessions and the generalized current session
- Predicting candidate classes

Our *Customizable* Framework (9)



Our *Customizable* Framework (10)

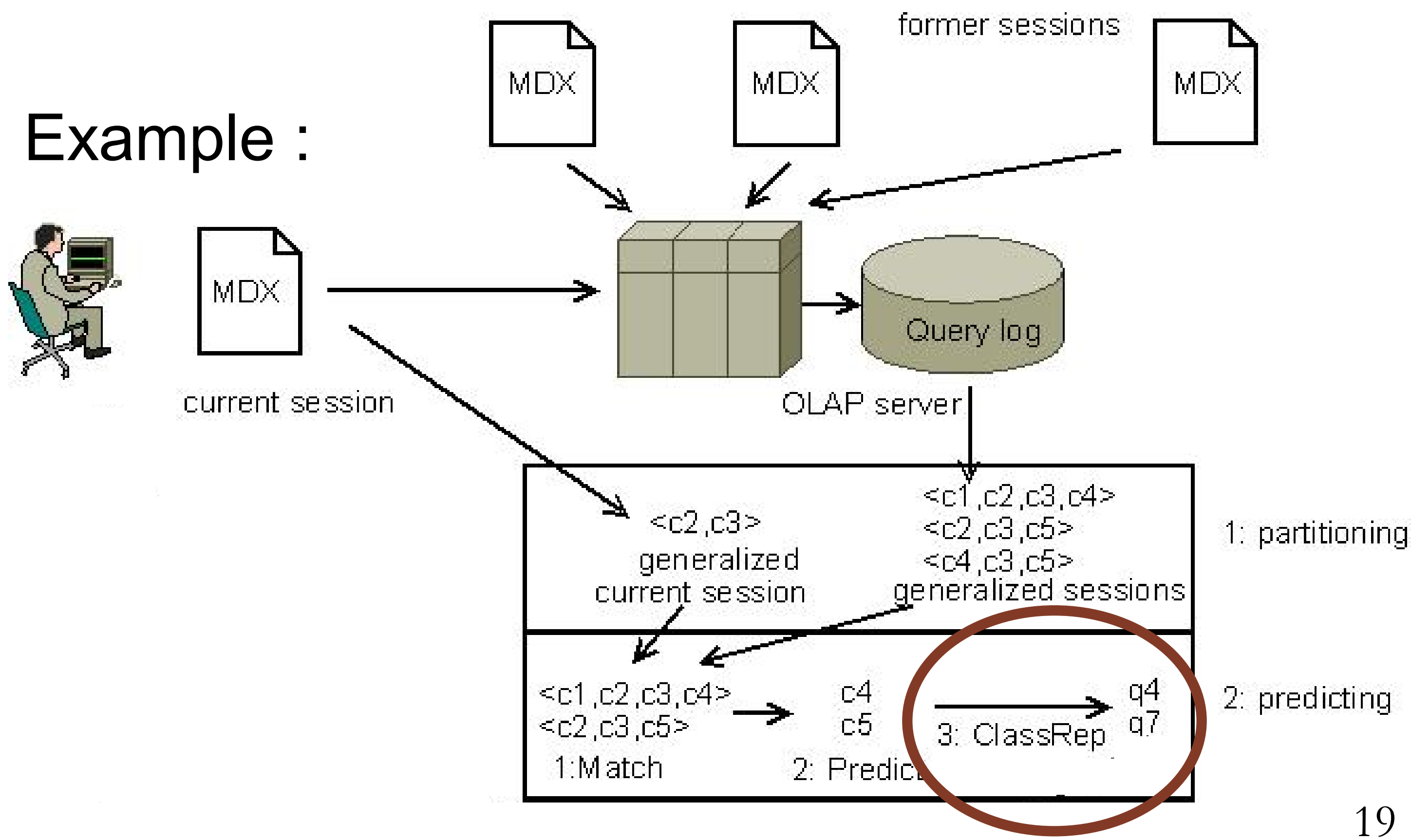
6 parameterized steps

- Partitioning the log
- Generalizing the sessions
- Matching generalized sessions and the generalized current session
- Predicting candidate classes
- Obtaining candidate recommendations

Our *Customizable* Framework (11)

- **Class Representing function:**
  Medoid of the candidate class
  ➡ *A set of candidate recommendations*

- Example :



Our *Customizable* Framework (12)

6 parameterized steps

- Partitioning the log
- Generalizing the sessions
- Matching generalized sessions and the generalized current session
- Predicting candidate classes
- Obtaining candidate recommendations
- Ranking candidate recommendations

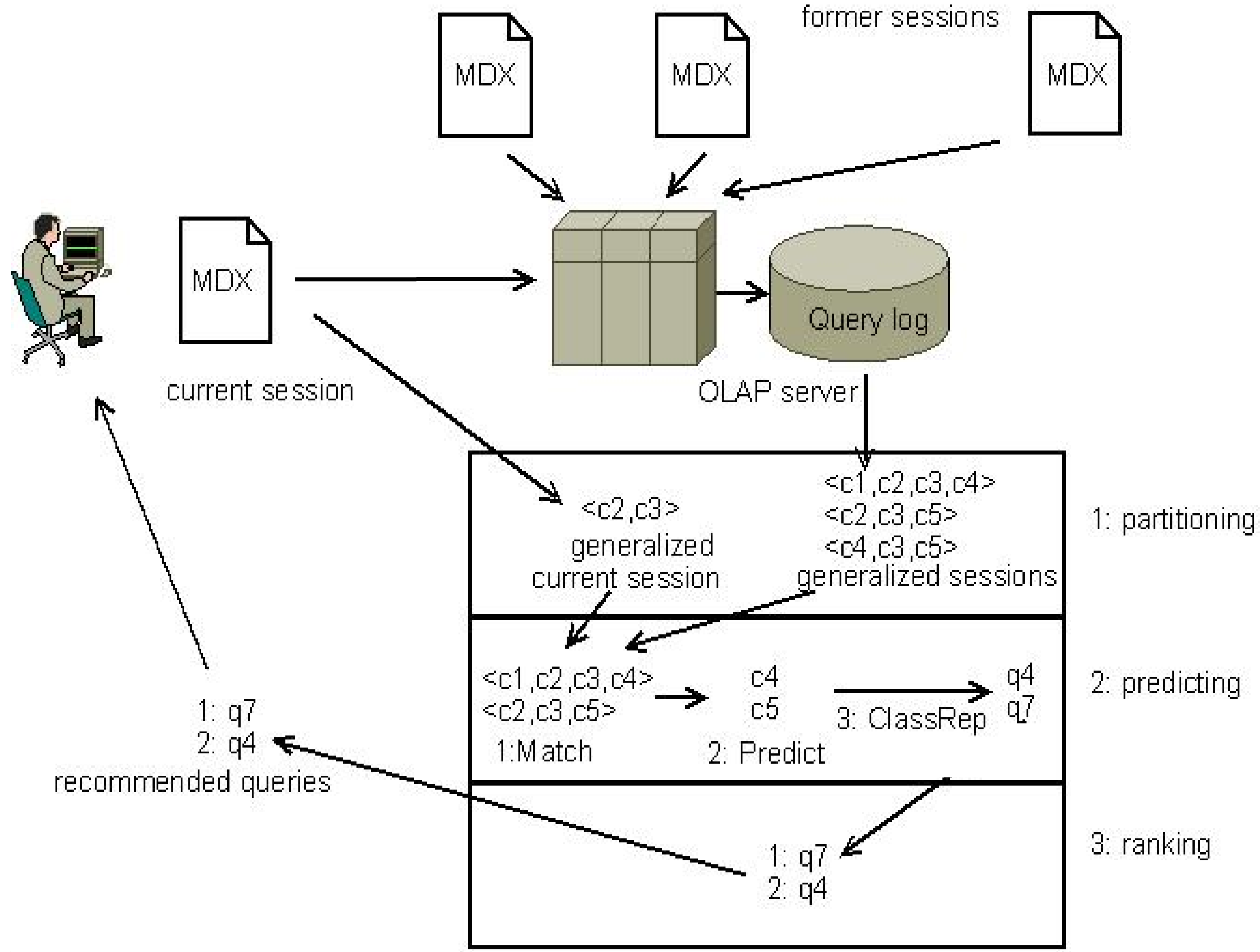
Our *Customizable* Framework (13)

1) What is the problem? Motivations, Intuitions and Example

2) How to solve the problem? Our framework

3) Experimentations

4) Conclusion and Future work



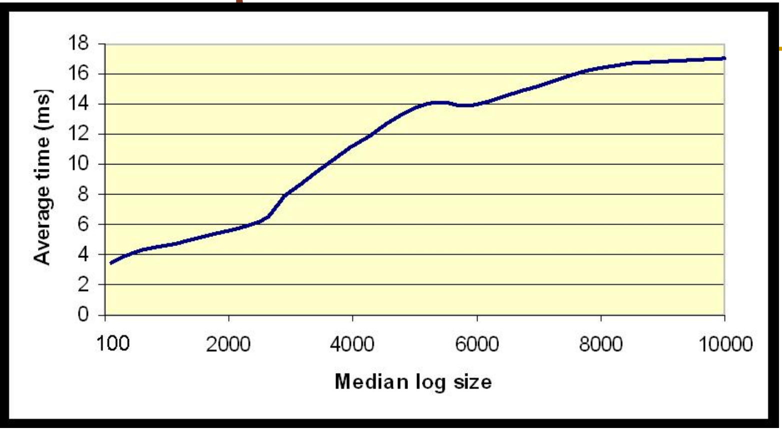
Experimentation (1)

<u>Our generator</u>

- The cube:
  - 6 dimensions
  - A maximum of 4 levels per dimension
  - A maximum of 100 values per dimension
    - *We obtain a cube of 1 000 000 000 000 references.*

- The sessions :
  - A maximum of 100 references per MDX query
  - X sessions in the log
  - Maximum Y queries per session

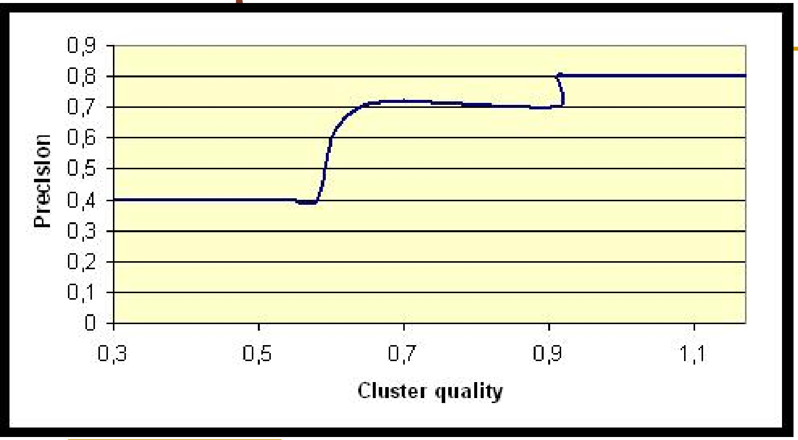
Experimentation (2) – Results - Performance



Performance analysis

- Measure of the time taken to propose a recommendation
  - 20 < X < 200 sessions
  - 10 < Y < 150 queries per session
  - 100 < log < 10 000 queries
  - Current session : 1 < Y < 100 queries
- Observations :
  - Time increases slowly with log size
  - Negligible time (<18ms)

Experimentation (3) – Results - Precision



Precision of the recommendation

- Measure of the proportion of perfectly matching sessions
  - Current session : one of the session of the log without its last query
  - Ideally, the recommendation is this last query…
- Observations :
  - Precision increases with cluster quality
  - Good precision from cluster quality = 0.7

Conclusion and Future work

- Contribution
  - Proposition of a customizable framework
  - One instantiation for MDX queries
  - Results of experiments:
    - Recommendations can be computed efficiently
    - Precise and objectively good recommendations

- Future work:
  - Experiments on real data sets with real users
  - Others instantiations of the framework
    - Compare instantiations
  - Pushing OLAP operations (roll-up, …) into the framework

Thank you for your attention.

Any questions ?

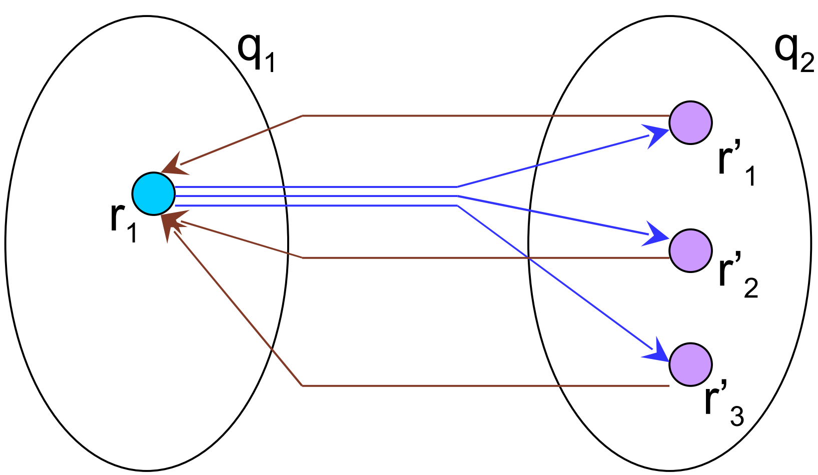
Hausdorff Distance

The Hamming Distance : $d(r_1, r_2) = d(\langle a_1, ..., a_N \rangle, \langle b_1, ..., b_N \rangle)$
$= \sum_{i=1}^{N} a_i = b_i |$ if $a_i = b_i$ then 0 else 1

The Hausdorff Distance :

$$d_{\mathrm{H}}(q_1, q_2) = \max\{\, \sup_{r_1 \in q_1} \inf_{r_2 \in q_2} d(r_1, r_2),\ \sup_{r_2 \in q_2} \inf_{r_1 \in q_1} d(r_1, r_2) \,\}$$

- Cube C= {Time, Vehicle, Customer, Garage, REPAIR}
  - q_1 = Total number of repairs in 2005 for the North region
    = {<2005, All, ALL, North>}
    = $\{r_1\}$
  - q_2 = Total number of repairs in 2005 for garages G1, G2 and in North region where the customer is Elsa
    = {<2005, All, Elsa, G1>, <2005, All, Elsa, G2>, <2005, All, Elsa, North>}
    = $\{r'_1\}\{r'_2\}\{r'_3\}$



Hausdorff Distance

The Hamming Distance : $d(r_1, r_2) = d(\langle a_1, ..., a_N\rangle, \langle b_1, ..., b_N\rangle) = \sum_{i=1}^{N} a_i = b_i|$ if $a_i = b_i$ then 0 else 1

The Hausdorff Distance :

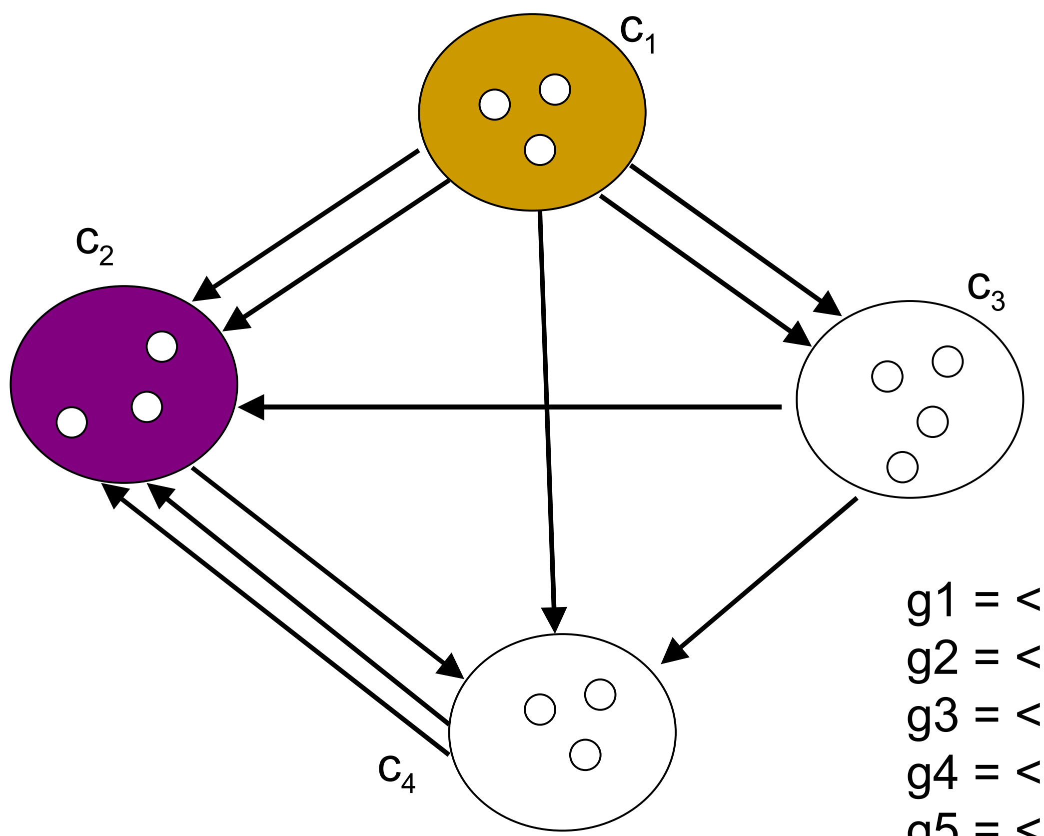
$$d_{\mathrm{H}}(q_1, q_2) = \max\{\sup_{r_1 \in q_1} \inf_{r_2 \in q_2} d(r_1, r_2), \sup_{r_2 \in q_2} \inf_{r_1 \in q_1} d(r_1, r_2)\}$$

- Cube C= {Time, Vehicle, Customer, Garage, REPAIR}
  - q1 = Total number of repairs in 2005 for the North region
    = {<2005, All, ALL, North>} = $\{r_1\}$
  - q2 = Total number of repairs in 2005 for garages G1, G2 and in North region where the customer is Elsa
    = {<2005, All, Elsa, G1>, <2005, All, Elsa, G2>, <2005, All, Elsa, North>} = $\{r'_1\}\{r'_2\}\{r'_3\}$
- Hamming Distance calculation:
  - $d(r_1, r'_1) = d(r'_1, r_1) = 0 + 0 + 1 + 1 = 2$
  - $d(r_1, r'_2) = d(r'_2, r_1) = 0 + 0 + 1 + 1 = 2$
  - $d(r_1, r'_3) = d(r'_3, r_1) = 0 + 0 + 1 + 0 = 1$
- Hausdorff Distance calculation:

$$\left.\begin{array}{l} \left.\begin{array}{ll} x_1 & = inf\{d(r_1, r'_1), d(r_1, r'_2), d(r_1, r'_3)\} \\ & = inf\{2, 2, 1\} = 1 \end{array}\right\} t_1 = sup\{x_1\} = 1 \\ \\ \left.\begin{array}{ll} x'_1 & = inf\{d(r'_1, r_1)\} = inf\{2\} = 2 \\ x'_2 & = inf\{d(r'_2, r_1)\} = inf\{2\} = 2 \\ x'_3 & = inf\{d(r'_3, r_1)\} = inf\{1\} = 1 \end{array}\right\} t_2 = sup\{x'_1, x'_2, x'_3\} = 2 \end{array}\right\} d_H = max\{t_1, t_2\} = 2$$

Hub and Authority

Authority , Hub :



g1 = <c1, c2, c4, c2>
g2 = <c1, c3, c2>
g3 = <c3, c4>
g4 = <c1, c4, c2>
g5 = <c1, c3>
g6 = <c1, c2>

Approximative String Matching

- Finding approximate matches to a pattern in a string
- Closeness of a match: number of primitive operations necessary to convert the string into an exact match.
- Usual primitive operations:
  - *insertion* (*cot* ⇨ *coat*),
  - *deletion* (*coat* ⇨ *cot*), and
  - *substitution* (*coat* ⇨ *cost*).
  - Possibly : *transposition (cost* ⇨ *cots)*