A Collaborative Filtering Algorithm with Clustering for Personalized Web Service Selection in Business Processes

Dionisis Margaris, Panagiotis Georgiadis and Costas Vassilakis

University of Athens, Greece
Department of Informatics and Telecommunications

Introduction

- Collaborative filtering exploits the known preferences of a group of users to formulate **recommendations** or predictions of the unknown preferences for other users**.**
- Collaborative filtering algorithms also handle complex items, which are described using hierarchical tree structures containing rich **semantic information** that must be taken into account in order to make accurate recommendations
- In contexts where items to be recommended are associated with QoS parameters (e.g. services implementing parts of business processes), collaborative filtering techniques must also take into account the items' **QoS parameters**, so as to generate recommendations tailored to the individual user needs.
- In order to support the efficient and scalable execution of collaborative filtering algorithm, **clustering techniques** can be used
    - Cluster formulation is typically performed in an offline fashion

Related Work

Personalized Web Service Selection in Business Processes

- ✓ An integrated framework for QoS-based adaptation and exception resolution in WS-BPEL scenarios (Margaris et al.)

  Optimized recommendation based on QoS attributes

- ✓ Adapting WS-BPEL scenario execution using collaborative filtering techniques (Margaris et al.)

  Optimized recommendation based on Collaborative Filtering

- ✓ An integrated framework for adapting WS-BPEL scenario execution using QoS and collaborative filtering techniques (Margaris et al.)

  Optimized recommendation based on QoS attributes and Collaborative Filtering

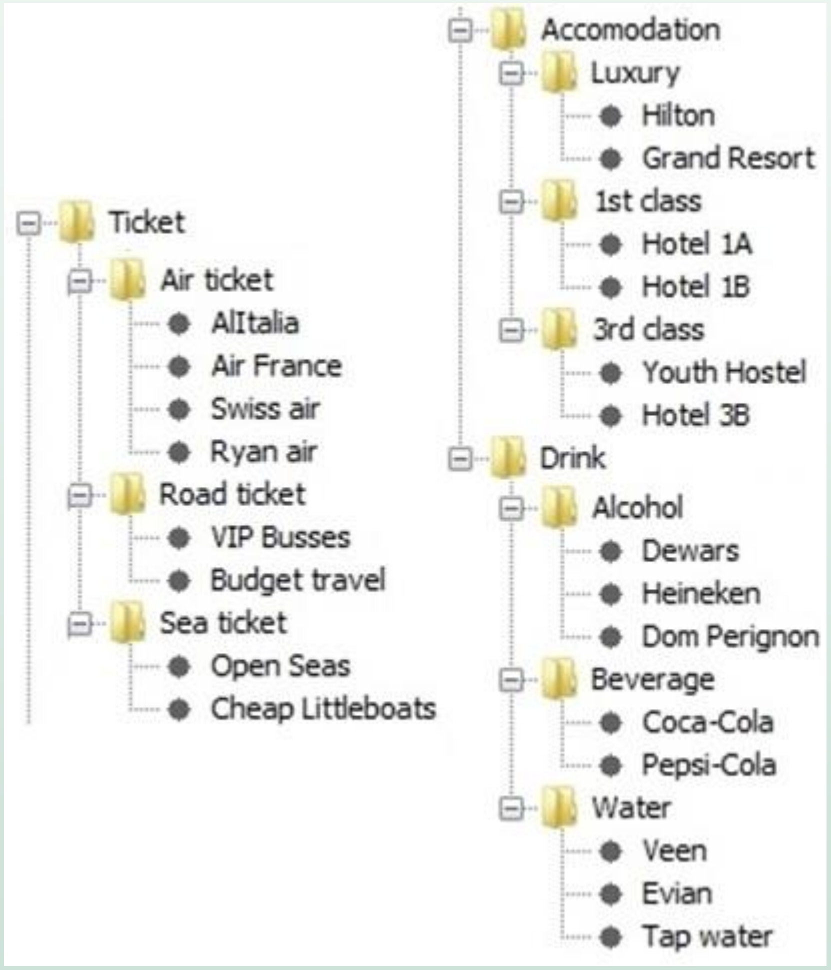
Clustering

- ✓ Adapting Finding Groups in Data: an Introduction to Cluster Analysis (Kaufman et al.)

  CLARA clustering algorithm and Silhouette coefficient

Service replacement candidates

- When a user requests a recommendation / adaptation for some service *A*, we can use any service offering *the same* or *more specific* functionality than A
  - For instance, if a user requests a "travel" service, we can use air travel, sea travel or land travel
  - However, if a user requests an "air travel" service, we can use any air travel service (including more specialized ones e.g. *helicopter travel*), but we cannot use sea travel
- To be able to determine the service replacement candidates, we use a tree structure representing service hierarchies



Service hierarchies (or Semantic WS trees)

QoS Aspects And Definitions

- Typical QoS attributes
  - **Cost, Response Time, Availability**, Reliability, Security, etc
- A business process invocation includes QoS specifications which may designate (for each QoS attribute):
  - An upper bound and a lower bound, specific to each task within the business process
  - A weight, which applies to *all* tasks within the business process

  The specifications are defined via vectors, i.e.:

    - **MAX(task$_i$)** = ($rt_{max}(task_i)$, $c_{max}(task_i)$, $av_{max}(task_i)$),
    - **MIN(task$_i$)** = ($rt_{min}(task_i)$, $c_{min}(task_i)$, $av_{min}(task_i)$),
    - **W** = (rt_w, c_w, av_w), a *single specification* applying to all invocations

In this work, QoS attributes are normalized in the range [0, 10] and it is arranged so that always “higher values are better”

- E.g. a service with cost=3 is *more expensive* than a service with cost=5

Collaborative filtering prerequisites (1/2)

- In collaborative filtering, we need metrics to quantify how “similar” two users are
- In our context, instead of users we consider past executions of the same business process:

# exec	Travel	Hotel	Drink
1	Swiss	Hilton	Heineken
2	Alitalia	Hilton	Heineken
3	Ryanair	Hotel_1a	Heineken
4	Alitalia	Youth_Hostel	Dom_Perignon
5	Ryanair	Youth_Hostel	Tap_Water
6	Budget_Travel	Hotel_3B	Tap_Water
7	Open_Seas		Evian

past executions repository

Collaborative filtering prerequisites (2/2)

- To compute the distance between two executions, we combine the distances between the individual services invoked in the context of the executions
  - The distance between the two services synthesizes the *semantic distance dimension* and the *QoS-based distance dimension*.
- Regarding the semantic dimension, we adopt the semantic similarity distance metric between two services proposed in “A semantic distance measure for matching web services” (Bramantoro et al.):

$$ssim(s_1,s_2) = (C - lw * PathLength - NumDownDirection) / C$$

- *C* is a constant set to 8
- *lw* is the level weight for each path within the service hierarchy tree, and depends on the depth of the tree.
- *PathLength* is the number of edges counted from service s_1 to service s_2 and
- *NumDownDirection* is the number of edges counted in the directed path between service s_1 and s_2 and whose direction is towards a lower tree level.

QoS aspects prerequisites (1/2)

Service	responseTime	cost	availability
Dewars	6	3	8
Heineken	7	8	7
Dom Perignon	6	1	9
Veen	5	2	9
Evian	8	5	8
Tap water	8	10	6
Hilton	7	2	7
Grand Resort	7	3	7
Youth_Hostel	5	9	5
Hotel_3B	5	8	5
Alitalia	8	7	4
AirFrance	8	6	9
Swiss	10	3	10
Ryanair	9	9	3
VIP_Buses	7	3	7
Budget_Travel	6	9	7

QoS values within the repository

QoS aspects prerequisites (2/2)

- Regarding the QoS-based distance dimension, the distance between two services is computed using the Euclidean distance metric; in the computation, each QoS dimension is weighted using the QoS attribute weight specified for the current adaptation.
- The attributes values are normalized by dividing them with the maximum value of the attribute within the corresponding category, in order to reflect how close to the maximum value within the category the specific value is:

$$qdist(s_1, s_2) = \sqrt{\sum_{q \in \{cost, av, respTime\}} \left(\frac{q(s_1)}{q_{max}(cat(s_1))} - \frac{q(s_2)}{q_{max}(cat(s_2))} \right)^2 * w_q}$$

- *q(s_i)* denotes the value of QoS attribute *q* (c, av, rt) for service s_i,
- w_q is the weight assigned to QoS attribute *q*
- $q_{max}(cat(s_i))$ is the maximum value present in the repository regarding QoS attribute *q* under the category in which s_i is a direct child

$$qsim(s_1, s_2) = 1 - qdist(s_1, s_2).$$

Overall Similarity Metric

Combining the semantic similarity with the QoS-based similarity, we compute the overall similarity metric of two services which is:

$$sim(s_1, s_2) = ssim(s_1, s_2) * qsim(s_1, s_2)$$

The formula for computing the similarity between two past executions is then shaped as (modified Sorensen Similarity Metric):

$$similarity(pe_1, pe_2) = \frac{2 * \sum_i sim(s_{1,i}, s_{2,i})}{|pe_1| + |pe_2|}$$

The CF adaptation algorithm

(1) Formulating a scenario-level functionality vector $F=(f_1, f_2,\dots, f_n)$, where each f_i corresponds to a functionality that is part of the current scenario

(2) For each functionality $funct_i$(request) for which a recommendation is requested, the algorithm retrieves from the repository the rows (past scenario executions) that have invoked a WS belonging to this category (being either *the same node* or a *descendant* in the service hierarchy tree)

(3) The rows for which the QoS characteristics of service $funct_i$(row) do not satisfy the bounds set through vectors MIN($funct_i$) and MAX($funct_i$) are dropped

(4) For each row retained, we compute its similarity with the current request

(5) The algorithm retains only the K-nearest neighbors, it groups the retained rows by the value of the service implementing the $funct_i$(request) functionality and computes the sum of the scores within each group.

(6) The service corresponding to the group having the greatest sum is then selected to deliver the specific functionality in the context of the current execution.

Clustering

- A clustering technique is used for supporting the efficient and scalable execution of proposed algorithm under the presence of large repositories of sparse data
- The computation of the clusters is performed in an off-line fashion, and the clustered repository is made available to the recommendation algorithm as soon as the computation is complete; therefore, the performance of the clustering technique does not penalize the recommendation process
- The cluster computation method uses the CLARA clustering algorithm (“Adapting Finding Groups in Data: an Introduction to Cluster Analysis”, Kaufman et al.) to formulate clusters
- Since the number of clusters *K* that will deliver the optimal clustering performance is not however known a priori, the iterated local search paradigm (“Iterated Local Search”, Lourenco et al.) is used to reduce the search range for *K*, using the Silhouette coefficient (“Adapting Finding Groups in Data: an Introduction to Cluster Analysis”, Kaufman et al.) as a solution quality metric

The clustering algorithm

- The potential range of the optimal cluster number is determined as $[\frac{\sqrt{N/2}}{2}, 2 * \sqrt{N/2}]$ ("Multivariate Analysis", Mardia et al.)
- We then extract the initial starting points of an iterated local search procedure with logarithmic cardinality from the above range as follows:
- The distance between the starting points is set to $d = \log_{10}(2 * \sqrt{N/2} - \frac{\sqrt{N/2}}{2}) * 10$
- The set of initial starting points is set to $\{\frac{\sqrt{N/2}}{2} + \frac{d}{2}, \frac{\sqrt{N/2}}{2} + \frac{3d}{2}, \frac{\sqrt{N/2}}{2} + \frac{5 * d}{2}, \ldots, 2 * \sqrt{\frac{N}{2}} - \frac{d}{2}\}$
- A hill climbing algorithm is executed for each point.
- The clusterings that have been produced by the execution of each *hill climbing* procedure are collected, and the one having the greatest Silhouette coefficient value is chosen.

The service recommendation algorithm

1. The adaptation algorithm formulates a task vector T=(t_1, t_2, …, t_n), where each t_i corresponds to a task that is part of the business process

2. To retrieve the k-nearest neighbors (we have set *k=50* using the results from “Scalable collaborative filtering using cluster-based smoothing”, Xue et al.), the similarity of the task vector *T* with the cluster medoids (each one corresponds to a past execution) is initially computed.

3. The cluster with the highest similarity is selected and searched for past executions that fulfill the criterion

4. If less than 50 recommenders are found, the search continues to the remaining clusters, in descending order of similarity of the task vector *T* with the cluster medoids.

Example (1/4)

“ I want to stay at **Hilton** Hotel, order **Heineken** from room service and I want a **recommendation** for my **air ticket** booking.

The recommended service’s cost must be over 4 and the QoS weights are response time=10%, cost=70% and reliability=20% ”

The task vector is instantiated to *T=(Air Ticket, Hilton, Heineken).*

Example (2/4)

# exec	Travel	Hotel	Drink
1	~~Swiss~~	Hilton	Heineken
2	Alitalia	Hilton	Heineken
3	Ryanair	Hotel_1a	Heineken
4	Alitalia	Youth_Hostel	Dom_Perignon
5	Ryanair	Youth_Hostel	Tap_Water
~~6~~	~~Budget_Travel~~	~~Hotel_3B~~	~~Tap_Water~~
~~7~~	~~Open_Seas~~		~~Evian~~

Usage patterns repository

Service	rt	c	av
Dewars	6	3	8
Heineken	7	8	7
Dom Perignon	6	1	9
Veen	5	2	9
Evian	8	5	8
Tap water	8	10	6
Hilton	7	2	7
Grand Resort	7	3	7
Youth_Hostel	5	9	5
Hotel_3B	5	8	5
Alitalia	8	7	4
AirFrance	8	6	9
Swiss	10	3	10
Ryanair	9	9	3
VIP_Buses	7	3	7
Budget_Travel	6	9	7

Services' QoS values

Example (3/4)

$$\sum_i sim(T_i, row2_i) = \left(1 - \sqrt{\left(\frac{8.75}{10} - \frac{8}{10}\right)^2 * 0.1 + \left(\frac{6.25}{9} - \frac{7}{9}\right)^2 * 0.7 + \left(\frac{6.5}{10} - \frac{4}{10}\right)^2 * 0.2}\right) * \frac{8 - \frac{2}{3} * 1 - 1}{8}$$

$$+ \left(1 - \sqrt{\left(\frac{7}{8} - \frac{7}{8}\right)^2 * 0.1 + \left(\frac{2}{9} - \frac{2}{9}\right)^2 * 0.7 + \left(\frac{7}{7} - \frac{7}{7}\right)^2 * 0.2}\right) * \frac{8 - \frac{1}{3} * 0 - 0}{8}$$

$$+ \left(1 - \sqrt{\left(\frac{7}{8} - \frac{7}{8}\right)^2 * 0.1 + \left(\frac{8}{10} - \frac{8}{10}\right)^2 * 0.7 + \left(\frac{7}{9} - \frac{7}{9}\right)^2 * 0.2}\right) * \frac{8 - \frac{1}{3} * 0 - 0}{8}$$

= 0.87 * 0.79 + 1.0 * 1.0 + 1.0 * 1.0 = 2.69

$$\sum_i sim(T_i, row3_i) = 1.95$$

$$\sum_i sim(T_i, row4_i) = 1.35$$

$$\sum_i sim(T_i, row5_i) = 1.39$$

Example (4/4)

The similarity metrics, computed via the modified Sørensen similarity index, between T and these rows are:

similarity(T, row2) = 2 * 2.69 / (3+3) = 0.896

similarity(T, row3) = 2 * 1.95 / (3+3) = 0.65

similarity(T, row4) = 2 *1.35/ (3+3) = 0.45

similarity(T, row5) = 2 *1.39/ (3+3) = 0.46

Rows 2 and 4 form one group corresponding to service *Alitalia* and achieving an overall score of 1.346.

Rows 3 and 5 form a second group corresponding to service *Ryanair* with an overall score of 1.11.

Thus, service **Alitalia** is selected to realize the *AirTravel* task in the context of the current scenario execution.

Implementation and Performance

In the experiment we have varied the following parameters:

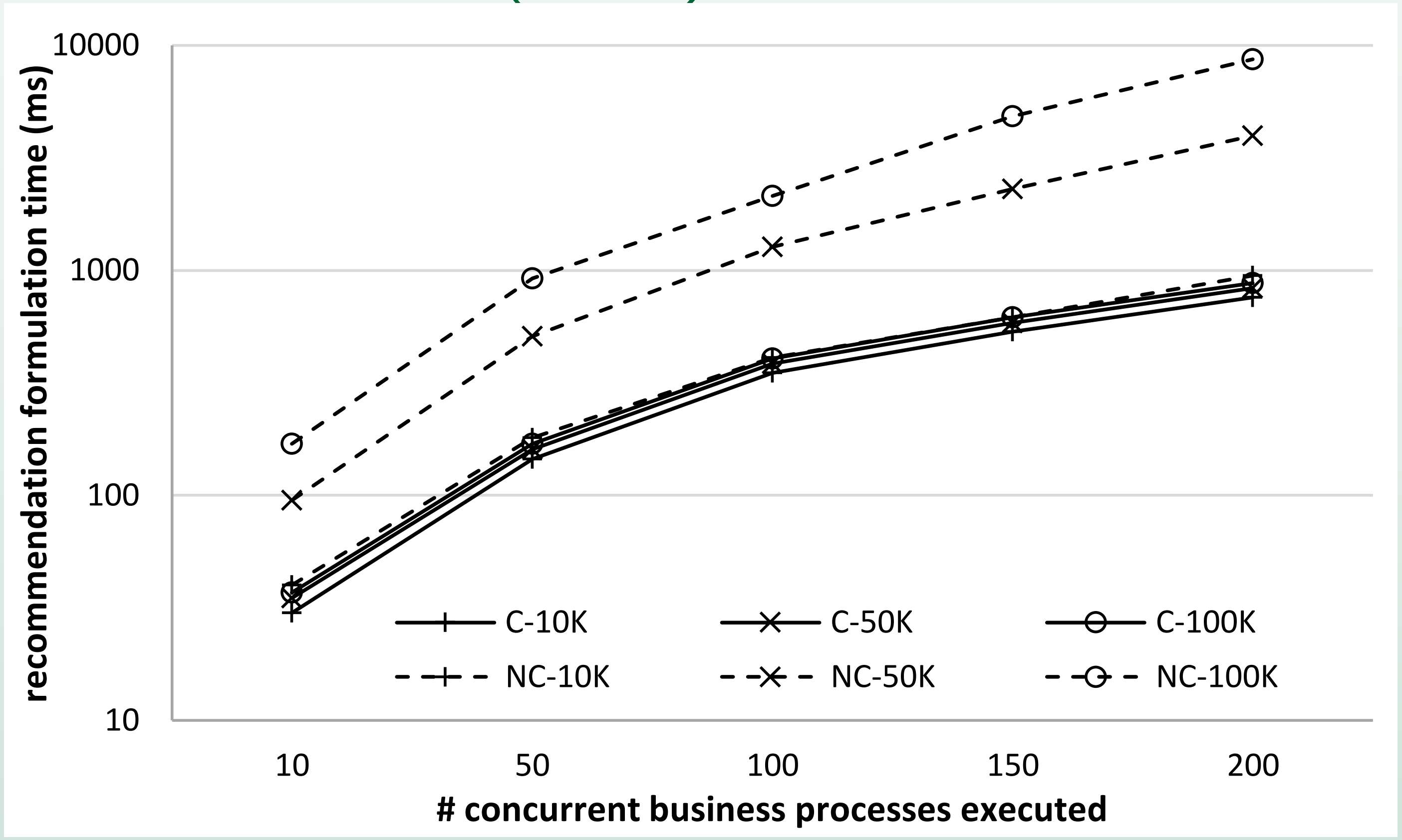
- the number of concurrent invocations
- the size of the past executions repository
- the number of functionalities in the scenario
- the number of recommendations requested

In all experiments, the semantic service repository was populated with synthetic data having an overall size of 2.000 web services, for 20 different tasks, with each task having 100 alternative providers.

The QoS attribute values in this repository were uniformly drawn from the domain [0,10].

Each unique performance evaluation test was run 100 times, and the average value was computed and is shown in the following diagrams.

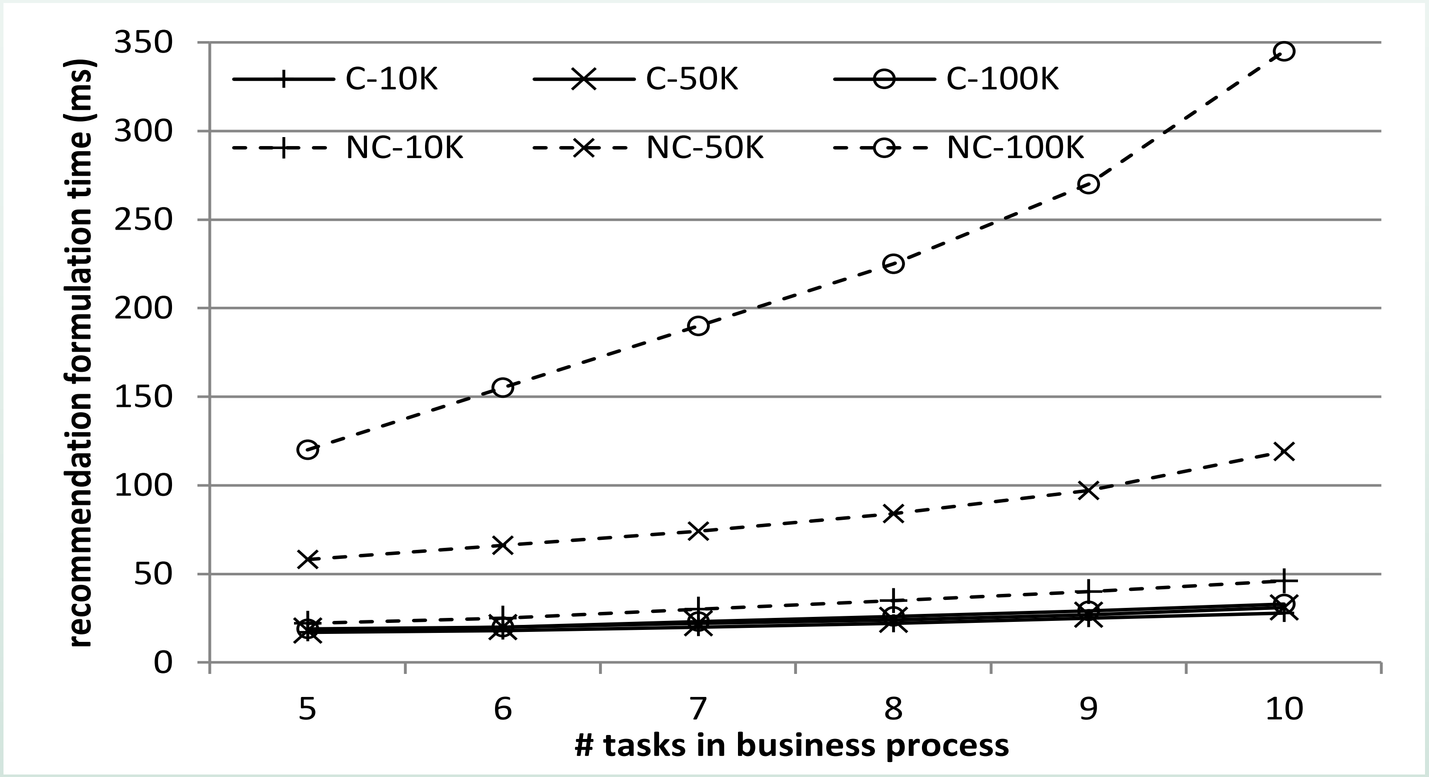
Performance (1/3)



Overhead imposed of the clustering (C) and the non-clustering (NC) algorithm respectively, under various concurrency level, when the past execution repository contains 10K, 50K and 100K entries.

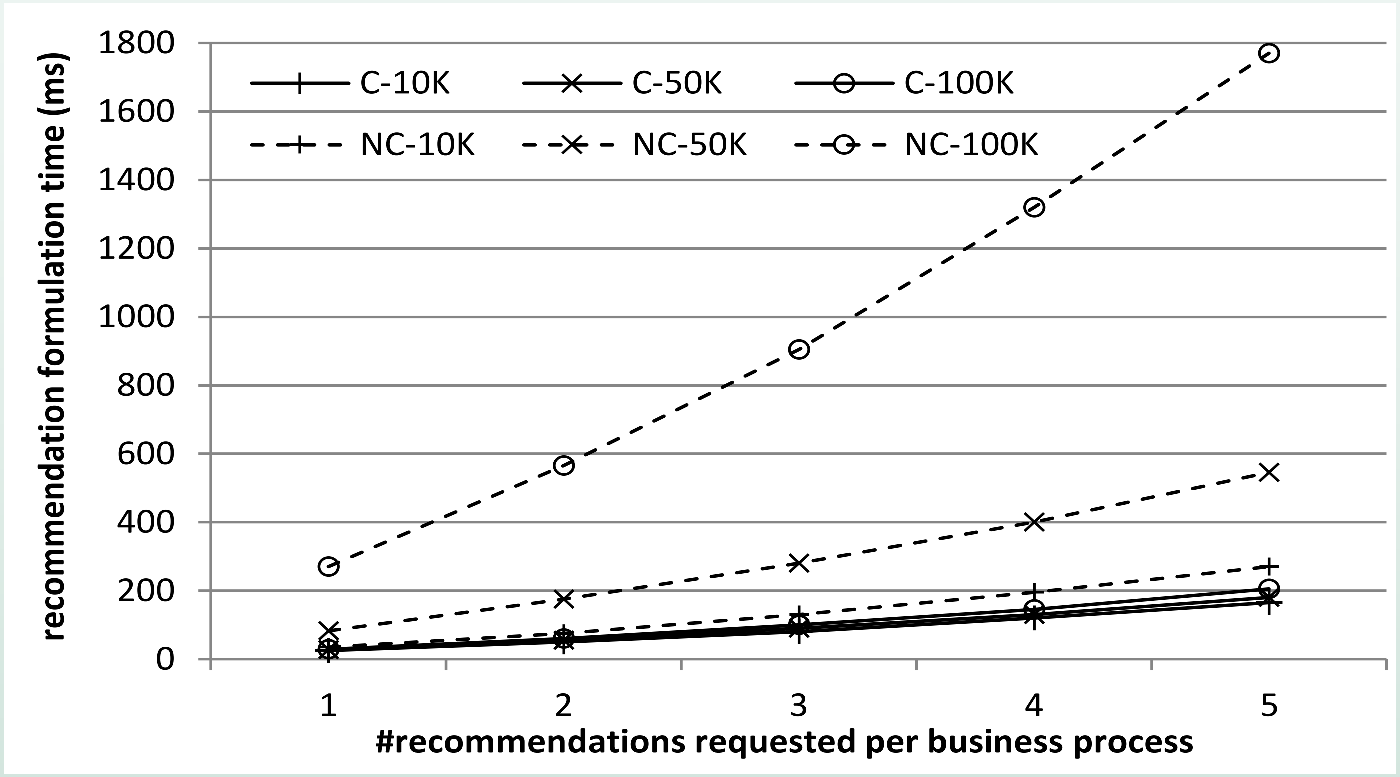
A business process with five tasks was used and one recommendation was requested, while the remaining four tasks were explicitly bound to specific service implementations.

Performance (2/3)



Overhead imposed to formulate the recommendation, when the number of tasks in the business process varies, of the clustering (C) and the non-clustering (NC) algorithm respectively, when the past execution repository contains 10K, 50K and 100K entries.
In these experiments, the concurrency level was set to one and for each business process a single recommendation was requested.

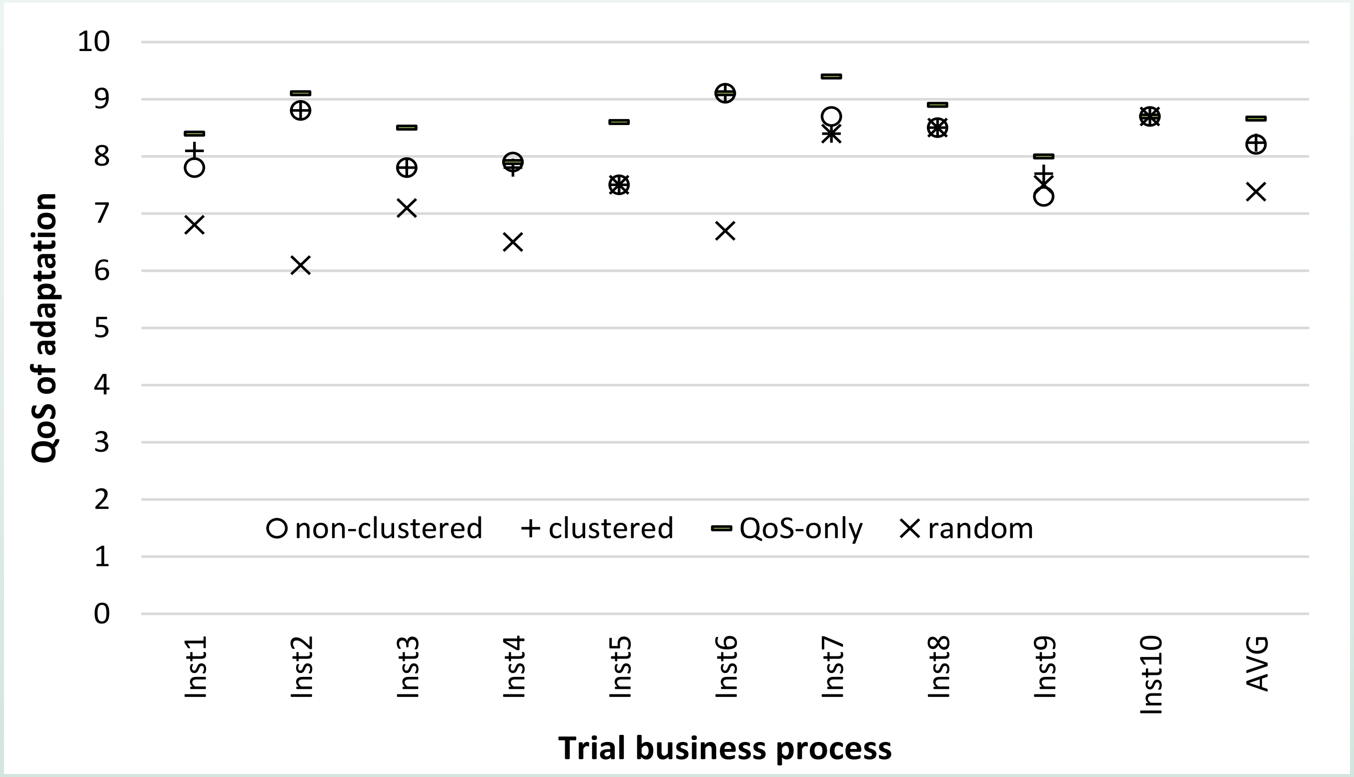
Performance (3/3)



Overhead imposed to formulate the recommendation, when the number of recommendations requested per business process varies, of the clustering (C) and the non-clustering (NC) algorithm respectively, when the past execution repository contains 10K, 50K and 100K entries.

In these experiments, a business process containing six tasks was used and the concurrency level was set to one.

Qualitative evaluation



The non-clustered algorithm produces recommendations having QoS in the range [87%, 100%] of the optimal ones (QoS-only) with an average equal to 93%, while the QoS of the clustered algorithm's recommendations fall in the range [84%, 100%] with an average of 90.9%.

The experiments show that the *precision at position k* metric for the algorithm ranges from 86% to 100% with an average of 94% (the clustering scheme achieves to retrieve, on average, 47 out of the 50 nearest neighbors to the current request).

Future Work

- Our future work will focus on considering incremental clustering techniques such as BIRCH (“BIRCH: an efficient data clustering method for very large databases”, Zhang et al.) etc.
  - Incremental clustering will remove the need to construct the clusters anew in order to accommodate the stream of new execution traces into the past executions repository.

- Investigate the effect of the *numlocal* and *maxneighbor* parameters of the CLARANS clustering algorithm. This will enable the replacement of CLARA by CLARANS, which is desirable since CLARANS is known to outperform CLARA both in execution time and cluster quality

- Additionally, we plan to conduct a user survey, in order to measure the degree to which users are satisfied by the recommendations generated by adaptation algorithm.

Thank you for your attention!

Questions ?

THANK YOU!