# *Rhone*: a quality-based query rewriting algorithm for data integration.

Daniel A. S. Carvalho
(Supervised by Chirine Ghedira-Guegan, Genoveva Vargas-Solar and Nadia Bennani, with inputs from Plácido A. Souza Neto)

Université Jean Moulin Lyon 3, Centre de Recherche Magellan, IAE, France
daniel.carvalho@univ-lyon3.fr

**Abstract.** In the traditional databases theory, data integration is a problem of merging data from data sources in order to provide a unified view of this data to the user. This PhD project addresses data integration in multi-cloud environments. Current data integration systems imply consuming data from data services deployed in cloud contexts and integrating the results. The project takes a new angle of the problem by proposing an SLA-based data integration approach in a multi-cloud environment which considers users' integration preferences (including quality constraints and data access requirements) to be matched with the services' quality constraints extracted from their SLAs while delivering the results. The objective is to enhance the quality and performance on data integration taking into consideration the economic model imposed by the cloud. At this stage, Our first results have shown that quality and performance can be enhanced, and the cost of the integration can be minimized by using our approach.

**Keywords:** Data integration. Query rewriting. Query rewriting algorithm. Cloud computing. SLA.

## 1 Introduction

Current data integration systems imply consuming and integrating data from data services which deliver data under different quality conditions related to freshness, trust, cost, reliability, availability, among others. Selecting data services, producing query rewritings and executing query plans are computationally expensive. These data integration tasks can take advantages from multi-cloud architectures considering their elasticity, *pay-per-use* model and parallel processing.

Multi-cloud environments bring new challenges to data integration due to different entities (data provider, data consumer, infrastructure, and the data) that should be taken into account, and their associated constraints and characteristics. Thus, given a user query, the integration process deals with different matching problems. They are: (i) matching the query and data services - the data

services have to produce a result for the query; (ii) matching the user preferences and the quality guarantees provided by the data provider - The preferences concern the data itself, the data services and the type of subscription the user has with the clouds; (iii) matching the user preferences and type of subscriptions - the subscriptions the user has with the clouds; and (iv) the data providers and the type of subscriptions - These subscription concern the ones data providers have with the clouds.

In cloud computing, the quality conditions that the user can expect from a service are defined in contracts called service level agreements (SLA). Current SLA models are not sufficient to cover the data integration requirements. Usually, SLAs are related to cloud resources, and not to the services. Thus, we strongly believe that SLAs can be re-modeled in order (i) to cover the limitations regarding the users' integration requirements (including quality constraints and data requirements); and (ii) to enhance the quality and performance in the current data integration solutions. Considering the aforementioned, the objectives of this PhD project contribute as following: (1) design of SLA model for data integration; (2) proposal of a data integration approach adapted to the vision of the economic model of the cloud. The originality of our approach consists in guiding the entire data integration solution - while selecting, filtering and composing cloud services, and delivering the results - taking into account (i) user preferences statements; (ii) SLA contracts exported by different cloud providers; and (iii) several QoS measures associated to data collections properties (for instance, trust, privacy, economic cost); and (3) validation of our approach in a multi-cloud scenario.

## 2 Related works

## 3 New vision of data integration

Figure 1 illustrates our vision of data integration. Data are accessed and retrieved as *data services* deployed in *cloud providers* geographically distributed. *Data services* and *cloud providers* export their SLA specifying the level of services they can guarantee to users. Two types of SLA are considered: (1) cloud SLA ($SLA_C$), agreements between a *data service* and a *cloud provider*; and (2) service SLA ($SLA_S$), agreements between *users* and *data services*. The figure 2 illustrates the cloud SLA schema used by our approach. The schema includes five groups describing (i) general information concerning the agreed service and parties; (ii) business rules (such as billing method, contract validity period, price, penalties and violations); and (iii) guarantees in terms of performance, security and data management.

Therefore, given a user query, his/her integration quality requirements and cloud subscription, the query is rewritten in terms of cloud services (*data services* and *data processing services*) composition that fulfill the integration requirements and deliver the expected results to the user. This new vision brings challenges to data integration, such as: (i) Performance issues. Given the huge

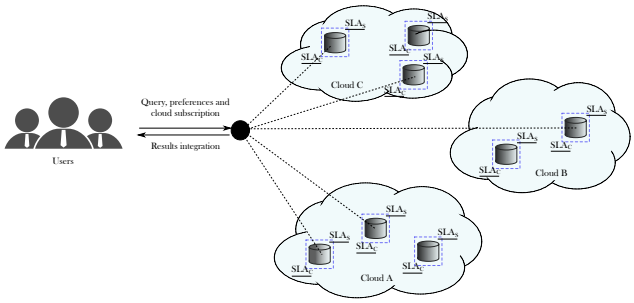


Fig. 1: Data integration scenario

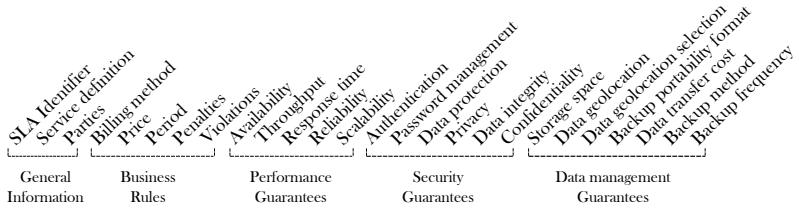


Fig. 2: Cloud SLA model

amount of *data services*, *data processing services* present in the multi-cloud context, the data integration tasks can require a huge amount of cloud resources and processing time; (ii) Economic model. Even with the on-demand and pay-per-use imposed by the cloud, users are limited to the resources they have contracted and to the budget they are ready to pay for while asking for a integration. (iii) Quality issues. Some rewritings produced and executed to the user query could not satisfy his/her quality requirements concerning privacy, data provenance, cost, among others. Consequently, users may become dissatisfied with the obtained results; (iv) SLA heterogeneity. In the multi-cloud, *cloud services* and *cloud providers* export their SLAs with different semantics and structure making the matching of users' integration requirements with the SLAs more challenging. Moreover, while integrating services, we can also face incompatibilities of SLAs such as measures with the same meaning but with different names, measures with the same name but with different associated values and units, among others; and (iv) Reuse. Rewriting and executing the user query is computationally costly in terms of processing time and economic cost. Thus, it is necessary to propose a manner of reusing previous integration in order to save time and money, but also meeting the user expectations.

Motivated by these challenges, our data integration approach adapted to the multi-cloud is detailed in the next sections.

### 3.1 SLA-based data integration

We propose our SLA-based data integration approach which includes (i) looking for services that can be used as *data services* and *data processing services*, the last are responsible to retrieve and integrate data; (ii) data retrieval and integration; and (iii) delivering results to the user considering context and resource consumption. In this sense, our approach is divided in four steps as illustrated in the figure 3.


Fig. 3: SLA-based data integration workflow.

Given a user *query*, a set of associated user *preferences*, *cloud providers* and *cloud services*:
**SLA derivation**. In this step, we compute what we call an *derived SLA* that matches user' integration requirements (including quality constraints and data requirements) with the SLA's provided by *cloud services*, given a specific user cloud subscription. The user may have general *preferences* depending on the context she wants to integrate her data such as economic cost, bandwidth limit, free services, and storage and processing limits. The *SLA derivation* is the big challenge while dealing with SLAs and particularly for adding quality dimensions to data integration. Furthermore, the *derived SLA* guides the query evaluation, and the way results are computed and delivered.
**Filtering data services**. The *derived SLA* is used (i) to filter previous *integration SLA* derived for a similar request in order to reuse results; or (ii) to filter possible *cloud services* that can be used for answering the query.
**Query rewriting**. Given a set of *data services* that can potentially provide data for integrating the query result, a set of service compositions is generated according to the *derived SLA* and the agreed SLA of each *data services*.
**Integrating a query result**. The service compositions are executed with services from one or several clouds where the user has a subscription. The execution cost of service compositions must fulfill the *derived SLA*. The clouds resources needed by the user to execute the composition and how to use them is decided taking in consideration the economic cost determined by the data to be transferred, the number of external calls to services, data storage and delivery cost.

The algorithm 1 describes in general lines our approach which integrate data from multi-cloud environments considering quality aspects. As input, we consider (1) a user query $Q$; (2) a set of user preferences $P$; (3) a user cloud subscription $C$; (4) a set of previous *integrated SLA* $\mathcal{I}_{\mathcal{SLA}}$, obtained from a local register; (5) a set of *data services* $DS$; (6) a set of *data processing services* $DPS$; and (7) a set of *cloud providers* $CP$. As output, the integration results $IR$ is delivered to the user while fulfilling her preferences and cloud subscription.

The approach begins looking for previous *integrated SLAs* in our local registry. From each *integrated SLA* we verify if it matches the user' query and preferences (lines 2 - 6), and adding them to the set $pI$ (line 4). If previous *integrated SLA* matches (line 7), the one that partially matches with the user preferences is selected (line 8). Then, a new *integrated SLA* is computed for user query based on the previous *integrated SLA*, her preferences and cloud subscription (line 9). The service compositions produced to a previous query is reused based its *integrated SLA* ($sI$). Otherwise, a new *integrated SLA* is created to user request given the query, her preferences and cloud subscription (line 11). The SLA information of *DS* and *DPS* is extracted and included in the set of $\mathcal{S}$(line 12). This new *integrated SLA* is updated by calling the *Rhone* procedure (line 13). The *Rhone* is responsible of (i) selecting *data services* (in $DS$) and *data processing services* (in $DPS$) which satisfy the user preferences based on their SLAs; and (ii) producing service compositions that completely match with the query and fulfill the user preferences. Finally, the integration results ($IR$) are produced and delivered to the user considering her cloud subscription. Note that while executing compositions, it is necessary to satisfy the user requirements and cloud subscription, and also to dynamically update the *integrated SLA* adding the information of contracts that were established to allow the integration. Finally, the *integrated SLA* is stored in our registry to be used in a next query.

**Algorithm 1** - SLA-based data integration

**Input:** Query ($Q$), user preferences ($P$), user cloud subscriptions ($C$), a set of *data services* ($DS$) and *data processing services* ($DPS$) and a set of previous *integrated SLA* ($\mathcal{I}_{\mathcal{SLA}}$).

**Output:** Integration results delivered considering the user cloud subscription.

1: $pI \leftarrow \emptyset$
2: **for all** $SLA_i$ in $\mathcal{I}_{\mathcal{SLA}}$ **do**
3: **if** $macthes(Q, P, SLA_i)$ **then**
4: $pI \leftarrow pI \cup \{SLA_i\}$
5: **end if**
6: **end for**
7: **if** $pI \neq \emptyset$ **then**
8: $sI \leftarrow chooseBest(P, pI)$
9: $newSLA_i \leftarrow deriveSLA(sI, P, C)$
10: **else**
11: $newSLA_i \leftarrow deriveSLA(Q, P, C)$
12: $\mathcal{S} \leftarrow extract(DS, DPS)$
13: $newSLA_i \leftarrow updateSLA(Rhone(Q, \mathcal{S}))$
14: **end if**
15: $IR \leftarrow execute(newSLA_i)$
16: **return** $IR$
17: **end function**

### 3.2 SLA-based query rewriting algorithm

To serve a proof of concept to our approach, we develop a query rewriting algorithm which is guided by users' integration requirements and service level agreements exported by different data services and cloud providers. Researches have refered to data integration as a service composition problem in which given a query the objective is to lookup and compose data services that can contribute to produce a result. Currently, we have formalized and developed a rewriting algorithm that considers user preferences and services' quality aspects while selecting services and producing rewritings called *Rhone* (See algorithm 2, invoked in the algorithm 1, line 12). The *Rhone* algorithm consists in four macro steps:
**Selecting data services**. The *Rhone* looks for *data services* (line 2) that can contribute to answer the query while satisfying the user' preferences;
**Building candidate service descriptions**. A candidate service description (CSD) describes how a *data service* can contribute to answer the query. CSD maps a *data service* (including its variables) to the entire query or part of it. For all selected services, the *Rhone* tries to build CSDs when the mapping of variables is possible (line 3).
**Combining CSDs**. All CSDs are combined (line 4) producing a list of CSD combinations. Each combination can be a rewriting of the query.
**Producing rewritings**. Each list of CSDs is checked in order to verify whether it is a rewriting of the query or not (line 9). To be a valid rewriting, it is forbidden to have CSDs covering the same part of the query. Rewritings are produced while the user' requirements are being respected (lines 6-14). The originality of our algorithm concerns the functions $\mathcal{T}_{\text{init}}[\![\ ]\!]$, $\mathcal{T}_{\text{cond}}[\![\ ]\!]$ and $\mathcal{T}_{\text{inc}}[\![\ ]\!]$. They are responsible to initialize, check and increment user preferences associated to some aggregation like user constraints (total cost). This means that these preferences are initialized (line 6), and for each element in the CSD list they are incremented (line 11), and rewritings are produced while they are respected (line 8). The result of this step is the list of valid rewritings of the query (line 14).

## 4 Preliminary results

We have developed a prototype of our query rewriting algorithm which takes into consideration users' requirements and services' quality aspects extracted from SLAs. The *Rhone* prototype is implemented in Java.

Currently, our approach runs in a local controlled environment simulating a mono-cloud including a service registry of 100 concrete services. Experiments were produced to analyze the algorithm's behavior concerning performance, and quality and cost of the integration. The figures 4 and 5 summarize our first results.

The experiments include two different approaches: (i) the *traditional approach* in which user preferences and SLAs are not considered; and (ii) the *preference-guided approach* (P-GA) which considers the users' integration requirements and SLAs.

**Algorithm 2** - *Rhone*

**Input:** A query $Q$, a set of user preferences, and a set of concrete services $\mathcal{S}$.
**Output:** A set of rewritings $R$ that matches with the query and fulfill the user preferences.

**function** $Rhone(Q, \mathcal{S})$
$\mathcal{L}_{\mathcal{S}} \leftarrow SelectCandidateServices(Q, \mathcal{S})$
$\mathcal{L}_{\mathcal{CSD}} \leftarrow CreateCSDs(Q, \mathcal{L}_{\mathcal{S}})$
$I \leftarrow CombineCSDs(\mathcal{L}_{\mathcal{CSD}})$
$R \leftarrow \emptyset$
$\mathcal{T}_{\text{init}}[\![\ \mathcal{A}gg(Q)\ ]\!]$
$p \leftarrow I.next()$
**while** $p \neq \emptyset$ **and** $\mathcal{T}_{\text{cond}}[\![\ \mathcal{A}gg(Q)\ ]\!]$ **do**
  **if** $isRewriting(Q, p)$ **then**
    $R \leftarrow R \cup Rewriting(p)$
    $\mathcal{T}_{\text{inc}}[\![\ \mathcal{A}gg(Q)\ ]\!]$
  **end if**
  $p \leftarrow I.next()$
**end while**
**return** $R$
**end function**

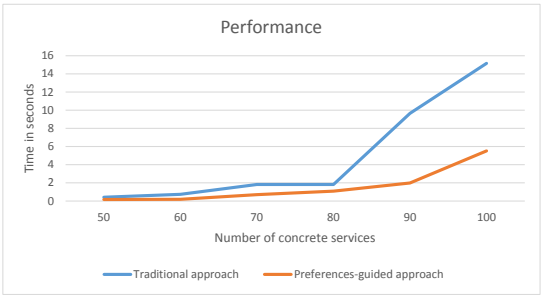


Fig. 4: Performance evaluation.

The results P-GA are promisingly. The *Rhone* increases performance reducing rewriting number which allows to go straightforward to the rewriting solutions that are satisfactory avoiding any further backtrack and thus reducing successful integration time (Figure 4). Moreover, using the P-GA to meet the user preferences, the quality of the rewritings produced has been enhanced and the integration economic cost has considerable reduced while delivering the expected results (Figure 5). However, the *Rhone* still need to be tested in a large scale case and in a context of parallel multi-tenant to test efficacy.

## 5 Conclusions and Research plan

This paper introduces a new vision of data integration adapted to the cloud context and to the end-user pay-per-use economic model. It also proposes a new

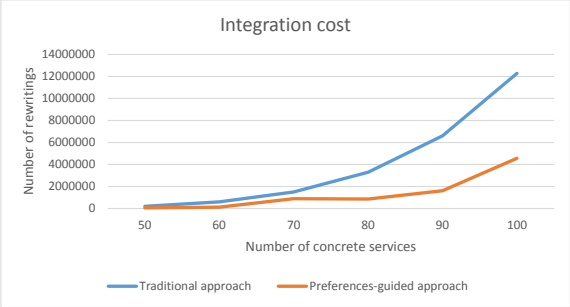


Fig. 5: Performance evaluation.

approach for data integration based on user requirements and SLA. In addition, a query rewriting algorithm called *Rhone* serves as proof for the feasibility of data integration process guided by cloud constraints and user preferences . Our first results are promisingly. The *Rhone* reduces the rewriting number and processing time while considering user preferences and services' quality aspects extracted from SLAs to guide the service selection and rewriting. Furthermore, the integration quality is enhanced, and it is adapted to cloud economic model reducing the total cost of the integration.

SLA incompatibilities are not treated in this paper. Currently we are working on this issue, and improving our SLA model and schema for data integration adapted to the multi-cloud context. Another important part is how to make efficient the rewriting process by reducing the composition search space. Finally, how should be a parallel execution of the query plan to let the execution efficient in the multi-cloud. In addition, we are focusing on evaluating and validating the entire quality-based data integration approach on a multi-cloud environment.

## References