ISSN 2395-8618

Optimizing Data Processing Service Compositions Using SLA's

Genoveva Vargas-Solar, José-Luis Zechinelli-Martini, and Javier-Alfonso Espinosa-Oviedo

***Abstract*—This paper proposes an approach for optimally accessing data by coordinating services according to Service Level Agreements (SLA) for answering queries. We assume that services produce spatio-temporal data through Application Programming Interfaces (API's). Services produce data periodically and in batch. Assuming that there is no full-fledged DBMS providing data management functions, query evaluation (continuous, recurrent or batch) is done through reliable service coordinations guided by SLAs. Service coordinations are optimized for reducing economic, energy and time costs.**

***Index Terms*—Data service, query optimization, workflow, service composition, SLA.**

I. Introduction

PERVASIVE denotes something "spreading throughout," thus a pervasive computing environment is the one that is spread throughout anytime anywhere and at any moment. From the computing science point of view what is interesting to analyze is how computing and software resources are available and provide services that can be accessed by different devices. For facilitating availability to these resources, they are wrapped under the same representation called service. A service is a resource handled by a provider, that exports an application programming interface (API). The API defines a set of method headers using an interface definition language. Consider a scenario where multiple users evolve within an urban area carrying GPS-enabled mobile devices that periodically transmit their location. For instance, the users location is notified by a stream data service with the following (simplified) interface:

```
subscribe() → {location:⟨nickname, coor⟩}
```

consisting of a subscription operation that, after invocation, will produce a stream of `location` tuples, each with a nickname that identifies the user and her coordinates. A stream is a continuous (and possibly infinite) sequence of tuples ordered in time.

The rest of the data is produced by the next two on-demand data services, each represented by a single operation:

```
profile(nickname) → {person:⟨age, gender,
                                email⟩}
interests(nickname) → {s_tag:⟨tag, score⟩}
```

The first provides a single `person` tuple denoting a profile of the user, once given a request represented by her nickname. The second produces, given the nickname as well, a list of `s_tag` tuples, each with a tag or keyword denoting a particular interest of the user (e.g. music, sports, fashion, etc.) and a score indicating the corresponding degree of interest.

Users access available services for answering some requirement expressed as a query. For instance, assume that Bob needs to find friends to make decisions whether he can meet somebody downtown to attend an art exposition. The query can be the following:

```
Find friends who are no more than
3 km away from me,
who are over 21 years old
and that are interested in art
```

But issuing the query from a mobile device, is not enough for evaluating it, some Service Level Agreements (SLA) need to also be expressed. For example, Bob wants the query to be executed as soon as possible, minimizing the battery consumption and preferring free data services.

Of course, the query cannot be solved by one service, some information will come for Google maps and Google location, other by Bob's personal directory, the availability of Bob's friend in their public agendas. Thus, the invocation to the different services must be coordinated by a program or script that will then be executed by an execution service that can be deployed locally on the user device or not. In order to do so, other key infrastructure services play an important role particularly for fulfilling SLA requirements. The communication service is maybe the most important one because it will make decisions on the data and invocation transmission strategies that will impact SLA.

Focussing on the infrastructure that makes it possible to execute the services coordination by making decisions on the best way to execute it according to given SLAs, we identify two main challenges:

– Enable the reliable coordination of services (infrastructure, platform and, data management) for answering queries.

Manuscript received on March 04, 2016, accepted for publication on June 15, 2016, published on June 25, 2016.

G. Vargas-Solar is Senior Scientist of the French Council of Scientific Research, LIG-LAFMIA Labs, France (web: http://www.vargas-solar.com).

J.L. Zechinelli Martini is a scientist at the LAFMIA Lab, France, – UDLAP Antena.

J.A. Espinosa-Oviedo is a scientist at the Barcelona Super Computer Centre, Spain, and a member of the LAFMIA lab, France.

ISSN 2395-8618

– Deliver request results in an inexpensive, reliable, and efficient manner despite the devices, resources availability and the volume of data transmitted and processed.

Research on query processing is still promising given the explosion of huge amounts of data largely distributed and produced by different means (sensors, devices, networks, analysis processes), and the requirements to query them to have the right information, at the right place, at the right moment. This challenge implies composing services available in dynamic environments and integrating this notion into query processing techniques. Existing techniques do not tackle at the same time classic, mobile and continuous queries by composing services that are (push/pull, static and nomad) data providers.

Our research addresses novel challenges on data/services querying that go beyond existing results for efficiently exploiting data stemming from many different sources in dynamic environments. Coupling together services, data and streams with query processing considering dynamic environments and SLA issues is an important challenge in the database community that is partially addressed by some works. Having studied the problem in a general perspective led to the identification of theoretical and technical problems and to important and original contributions described as follows. Section II describes the phases of hybrid query evaluation, highlighting an algorithm that we propose for generating query workflows that implement hybrid queries expressed in the language HSQL that we proposed. Section III describes the optimization of hybrid queries based on service level agreement (SLA) contracts. Section V discusses related work and puts in perspective our work with existing approaches. Section VI concludes the paper and discusses future work.

II. Hybrid Query Evaluation

We consider queries issued against data services, i.e., services that make it possible to access different kinds of data. Several kinds of useful information can be obtained by evaluating queries over data services. In turn, the evaluation of these queries depends on our ability to perform various data processing tasks. For example, data correlation (e.g. relate the profile of a user with his/her interests) or data filtering (e.g. select users above a certain age). We must also take into consideration restrictions on the data, such as temporality (e.g. users logged-in within the last 60 minutes).

We denote by "hybrid queries" our vision of queries over dynamic environments, i.e. queries that can be mobile, continuous and evaluated on top of push/pull static or nomad services. For example, in a mobile application scenario, a user, say Mike, may want to *Find friends who are no more than 3 km away from him, who are over 21 years old and that are interested in art*. This query involves three data services methods defined above. It is highly desirable that such query can be expressed formally through a declarative language named Hybrid Service Query Language (HSQL)[1]. With this goal in mind we adopt a SQL-like query language which is similar to CQL [1], to express the query as follows:

```
SELECT p.nickname, p.age, p.sex, p.email
FROM profile p, location l [range 10 min],
     interests i
WHERE p.age >= 21 AND
     l.nickname = p.nickname AND
     i.nickname = p.nickname AND
      'art' in i.s_tag.tag
      AND dist(l.coor, mycoor) <= 3000
```

The conditions in the `WHERE` clause enable to correlate `profile`, `location`, and `interests` of the users by their nickname, effectively specifying join operations between them. Additional conditions are specified to filter the data. Thus, users who are older than 21 and whose list of interests includes the tag `'art'`, and whose location lies within the specified limit are selected. For the location condition, we rely on a special function `dist` to evaluate the distance between two geographic points corresponding to the location of users, the current location of the user issuing the query is specified as `mycoor`. Since a list of scored tags is used to represent the interests of an user, we use a special `in` operator to determine if the tag `'art'` is contained in the list, while the list in question is accessed via a path expression.

Since the location of the users is subject to change and delivered as a continuous stream, it is neither feasible nor desirable to process all of the location data, therefore temporal constraints must be added. Consequently, the `location` stream in the `FROM` clause is bounded by a time-based window which will consider only the data received within the last 10 minutes. Given that the query is continuous, this result will be updated as the users' location changes and new data arrives. This is facilitated by a special `sign` attribute added to each tuple of the result stream, which denotes whether the tuple is added to the result (positive sign) or removed from it (negative sign).

In order to evaluate a declarative hybrid query like the one presented in the example we need to derive an executable representation of it. Such executable representation in our approach is a query workflow.

A. Query Workflow

A workflow fundamentally enforces a certain order among various activities as required to carry out a particular task. The activities required to evaluate a hybrid query fall into two basic categories: data access and data processing. Both of these types of activities are organized in a workflow following a logical order determined by the query. The execution of each of the activities, in turn, is supported by a corresponding service; data

[1]This language was proposed in the PhD dissertation of V. Cuevas Vicenttin of University of Grenoble.

ISSN 2395-8618

access activities by data services and data processing activities by computation services.

Following our service-based approach, the workflow used to evaluate a hybrid query consists of the parallel and sequential coordination of data and computation services. For example, the workflow representation for the query in the example is depicted in Figure 1.

The data services are represented by parallelograms, whereas computation services are represented as rounded rectangles and correspond to traditional query operators such as join or selection. The arrows indicating sequential composition not only imply order dependencies among the activities but also data dependencies; in particular, tuples that need to be transmitted between the different activities that produce and consume them.

Since the workflow enabling the evaluation of a given hybrid query acts in fact as a service coordination (comprising data and computation services), we refer to it as *query workflow*. Evaluating a hybrid query from a given query coordination depends first on finding the adequate (data and computation) services, second on their invocation, and finally on their communication and interoperation.

B. Computing a Query Workflow Cost

In order to use SLA's to guide query workflows evaluation, it is necessary to propose a cost model that can be used to evaluate a query workflow cost. The query workflow cost is given by a combination of QoS measures associated to the service methods it calls[2], infrastructure services such as the network and the hosting device it uses. The cost model considered for query workflows is defined by a combination of three costs: execution time, monetary cost and battery consumption. These costs are computed by calculating the cost of activities of a query workflow.

Query workflow cost: is computed by aggregating its activities costs based on its structure. The aggregation is done by following a systematic reduction of the query workflow such as in [2], [3]. For each sequential or parallel coordination, the reduction aggregates the activities costs. For the nested coordinations, the algorithm is applied recursively. The resulting cost is computed by a pondered average function of the three values.

$$\text{Cost} = \alpha\ (\text{temporal}_c) + \beta\ (\text{economic}_c) + \gamma\ (\text{energy}_c)\ /\ 3$$

We assume that the activity costs are estimated according to the way data are produced by the service (i.e., batch for on-demand services, continuous by continuous services).

Cost of an activity calling an on-demand service: For data produced in batch by on-demand services, the global activity cost is defined by the combination of three costs:

- *Temporal cost* given by (i) the speed of the network that yields transfer time consumption, determined by the data size and the network's conditions (i.e., latency and throughput) both for sending the invocation with its input data and receiving results; (ii) the execution of the invoked method has an associated approximated method response time which depends on the method throughput[3].
- *Economic cost* given by (i) the type of network: indeed, transmitting data can add a monetary cost (e.g., 3G cost for mega octets transfer); (ii) the cost of receiving results from a service method invocation sent to a specific service provider, for example getting the scheduled activities from the public agenda of my Friends can have a cost related to a subscription fee.
- *Energy cost* produced as a result of using the network and computing resources in the device hosting the service provider that will execute the method called. These operations consume battery entailing an energy cost.

Cost of an activity calling a continuous service cost: For data produced by continuous services the global activity cost is defined by the combination of three costs that depend on the data production rate resulting from the invocation of a method. The costs are multiplied by the number of times data must be pulled and transferred. The economic cost can be associated to a subscription model where the cost is determined by the production rate. For example receiving data frequently (e.g., give my current position every five minutes, where five minutes is the expected production rate) can be more expensive than receiving data in specific moments (e.g., the number of times Bob went to the supermarket during a month). The temporal and energy costs are also determined by the frequency in which data are processed (processing rate): data can be processed immediately, after a threshold defined by the number of tuples received, or the elapsed time, or a buffer capacity. Both production rate and processing rate impact the execution time cost, execution economic cost, and battery consumption cost.

III. Optimizing Hybrid Queries Using SLA Contracts

Given a hybrid query and a set of services that can be used for answering it, several query workflows can be used for implementing it. For example, consider the query workflow in Figure 2a that is a version of the friend finder example. The figure shows a query workflow coordinating activities in parallel for retrieving the profile and the location datasets. Then, the filtering activity that implements a window operator is placed just after the activity that retrieves the location. This activity reduces the input dataset size. Then, both datasets are correlated and finally the last activity filters the dataset to get data related only to 'Joe'. Placing the last activity just after of the retrieval of the profile dataset can reduce the processing time.

[2]QoS measures are a set of quantitative measures that describe the possible conditions in which a service method invocation is executed.

[3]The method throughput is given by the amount of requests in a period of time (e.g., each minute) and the state of the device such as memory or CPU.

ISSN 2395-8618

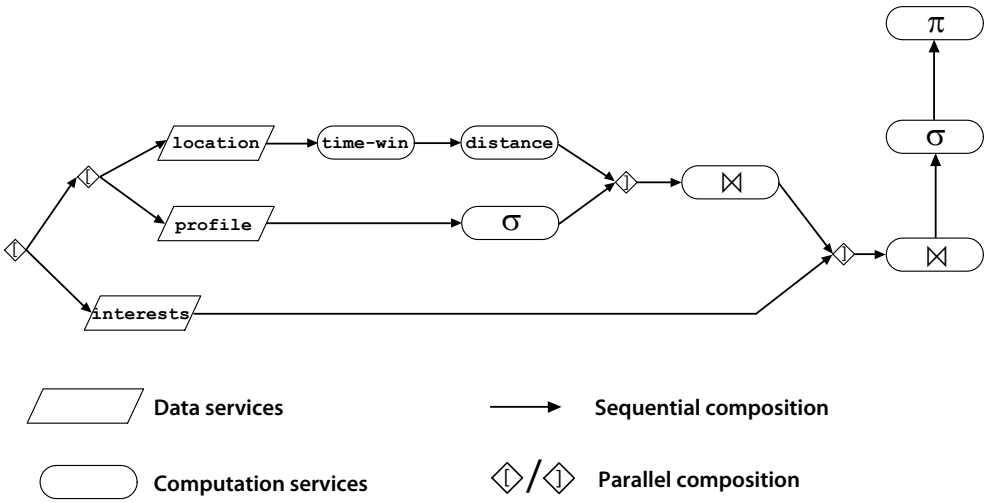


Fig. 1. Query coordination for the query the initial example

Now consider the query workflow in Figure 2b that coordinates activities sequentially. Each activity in the control flow consumes and produces data that at the end result in a dataset which is equivalent with the first one.

Optimizing a hybrid query implies choosing the query workflow that best implements it with respect to a given SLA. Similar to classic query optimization techniques, we propose an optimization process that consists in two phases: (i) generating its search space consisting in "all" the query workflows that implement a hybrid query and (ii) choosing the top-k query workflows with costs that are the closest to the SLA.

A. Generating Potential Query Workflow Space

We use rewriting operations (e.g. split, aggregate, parallelize, etc.) for generating a set of "semantically" equivalent query workflows The rewriting process is based on two notions: function and data dependency relationships.

- Function: represents a data processing operation. We consider the following functions:
  1) `fetch` for retrieving a dataset from a data provision service (e.g., get Bob's friends);
  2) `projection` of some of the attributes of each item (e.g., tuple) of a dataset (e.g., get name and location of Bob's friends assuming that there each friend has other attributes);
  3) `filter` the items of a dataset according to some criterion (e.g. Alice's friends located 3 Km from her current position); and,
  4) `correlation` of the items of two datasets according to some criterion (e.g., get friends shared by Bob and Alice that like "Art").
- Data dependency relationships between functions. Intuitively, given two functions with input parameters and an output of specific types, they are
  1) F_1 `independent` F_2 if they do not share input datasets;
  2) F_1 `concurrent` F_2 if they share common input datasets;
  3) F_1 `dependent` F_2 if they use common input datasets.

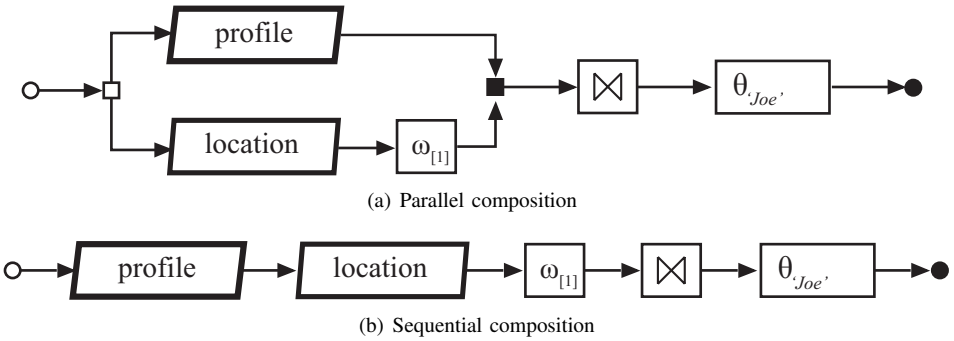
We propose rewriting rules and algorithms for generating a representation of an HSQL expression as a composite function and a data dependency tree. This intermediate representation is used for finally generating a query workflow search space. This generation is based on composition patterns that we propose for specifying how to compose the activities of a query workflow. Let F_1 and F_2 be functions of any type according to their dependency relationship they can give rise to two activities A_1 and A_2 related according to the composition patterns shown in Figure 3.

1) F_1 `independent` F_2 leads to three possible composition patterns: A_1 `sequence` A_2 or A_1 `sequence` A_2 or A_1 `parallel` A_2.
2) F_1 `concurrent` F_2 leads to the same sequential patterns of the previous case. In the case of the parallel pattern, it works only if and only if F_1 and F_2 are filtering functions or one of them is a filtering function and the other a correlation function.
3) F_1 `dependent` F_2 leads to a sequential composition pattern A_1 `sequence` A_2.

The search space generation algorithm ensures that the resulting query workflows are all deadlock free and that they terminate[4]. Once the search space has been generated, the query workflows are tagged with their associated three dimensional cost. Then, this space can be pruned in order to find the query workflows that best comply with the SLA

[4]Details on the algorithm can be found in the dissertation of Carlos Manuel López Enríquez of the University of Grenoble.

ISSN 2395-8618



(a) Parallel composition

(b) Sequential composition

Fig. 2. Query workflows of the *Search space*.

Relation	Notation	Composition pattern
A Independent of B	A||B	
A Concurrent with B	A⊳⊲B	
A Dependent on B	A⊳⊳B	

Fig. 3. Composition patterns

expressing the preferences of the user. This is done applying a top-k algorithm as discussed in the following section.

B. Computing an Optimization Objective

In order to determine which are the query workflows that answer the query respecting the SLA contract, we compute an optimization objective taking as input the SLA preferences and assuming that we know all the potential data and computing services that can be used for computing a query workflow. The SLA expressed as a combination of pondered measures, namely, execution time, monetary cost and energy. Therefore we propose an equation to compute a threshold value that represents the lowest cost that a given query can have given a set of services available and required for executing it and independently of the form of the query workflow (see Equation 1).

$$Opt(Q,R) = min(\sum_{j}(f(A_j,\Omega_j) - \gamma(Q,R_j))) \quad (1)$$

The objective is to find the combination of resources (R i.e., services) that satisfies a set of requirements (Q, i.e., the preferences expressed by the user and associated to a query). Every service participating in the execution of a query exports information about its available resources and used resources. For example the number of requests that a service can handle and the number of requests that are currently being processes. The principle of the strategy is described as follows: determine to which extent the required resources by Q can be fulfilled by a the resources provided by each service. The total result represents the combination of resources provided by available services that minimize the use of the global available resources (A) and the resources currently being used (Ω).

As shown in Figure 4 this value can be represented as a point in an n dimensional space where each dimension represents a SLA measure. Similarly, as shown in Figure 4, the query workflows cost which is also defined as a function of these dimensions, can be represented as a point in such n-dimensional space. The optimization process looks for points that are closest to the objective point by computing the Euclidean distance.

C. Choosing an Optimum Plan Family

We adopt a top-k algorithm in order to decide which of the k query workflows that represent the best alternative to implement a hybrid query for a given SLA. We adopt the Fagin's algorithm [4]. The top-k algorithm assumes m inverted lists $L_1, \ldots, L_m$ each of which is of the form $[\ldots, (qw_i, c_{i,j}), \ldots]$ with size $|\mathcal{S}|$ where $i \in [1 \ldots |\mathcal{S}|]$, and $j \in [i \ldots m]$. The order of the inverted lists depends on the algorithm.

The Fagin algorithm assumes that each list L_j is ordered by $c_{i,j}$ in ascending order. The principle is that the k query workflows are close to the top of the m lists. In the worst case, the $c_{k,j}$ is the last item for some $j \in [1, \ldots, m]$. The algorithm traverses in parallel the m lists by performing sequential access. Once k items have been visited in all the m lists, it performs random access over the m lists by looking for the already visited items and computes their scores. The scores are arranged in a sorted list in ascending order and thus the first k items are on the top of the score list. The main steps of the algorithm are:

1) Access in parallel the m lists by performing sequential access.
2) Stop once k query workflows have been seen in the m lists.

ISSN 2395-8618

$$QWF^1(p_1^1, p_2^1, \ldots, p_m^1)$$
$$QWF^2(p_1^2, p_2^2, \ldots, p_m^2)$$
$$\vdots$$
$$QWF^n(p_1^n, p_2^n, \ldots, p_m^n)$$

$$\forall i : 1 \leq i \leq n$$
$$d(p^i, o) = \sqrt{\sum_{j=1}^{m} (p_j^i - o_j)}$$

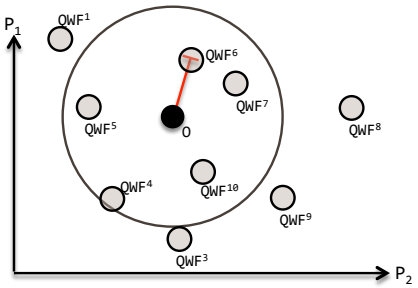


Fig. 4. Optimization approach

3) Perform random access over the m lists to obtain the m scaled attributes of each query workflow that has been processed and compute its $score$.
4) Sort in ascending order the scores.
5) Return the k query workflows on the top.

It is applied for obtaining an ordered family of query workflows that compose the optimum plan family. According to a descending order, the first query workflow will be executed. The rest of the queries can be stored as knowledge and they can be used for further optimizations. We do not consider learning based optimization [5] but we believe that such a technique can be applied in this case.

IV. Implementation and Experiments

We developed a proof of concept of our approach by implementing a service-based hybrid query processor named Hypatia for enacting query workflows. Figure 5 left side presents its architecture. The system is based on the Java platform. Queries in Hypatia are entered via a GUI and specified in our HSQL query language. Once a query is provided to the system it is parsed and then its corresponding query workflow is generated according to the algorithm that we described in Section III-A. The query parser and the query workflow constructor components perform these tasks. The parser was developed using the ANTLR[5] parser generator. The GUI also enables the user to visualize the query workflow, as well as the sub-workflows corresponding to composite computation services, which is facilitated by the use of the JGraph[6] library.

To implement stream data services we developed a special-purpose stream server framework, which can be extended to create stream data services from sources ranging from text files to devices and network sources. This framework employs Web Service standards to create subscription services for the streams. Stream data access operators in query workflows subscribe to these services and also receive the stream via special gateway services.

The evaluation of a query is enabled by two main components that support the computation services corresponding to data processing operations. A scheduler determines which service is executed at a given time according to a predefined policy. Composite computation services communicate via asynchronous queues and are executed by an ASM interpreter that implements our workflow model.

[5]http://www.antlr.org/
[6]http://www.jgraph.com/

A. Hybird Query Optimizer

We implemented a HYbrid Query OptimiZer (HyQoZ) that we integrated to the hybrid query evaluator Hypatia. Figure 5 shows the component diagram for processing hybrid queries with HyQoZ. An application representing a data consumer expresses hybrid queries based on the information about service instances provided by the APIDirectory, and define the SLA to fulfill. Applications may require either the evaluation of the hybrid query or the optimum query wrkflow implementing the hybrid query for its further execution. In such cases applications request either the evaluator Hypatiaor the optimizer HyQoZ.

Its components are described by REST interfaces and they exchange self-descriptive messages. The messages instantiate different coordinations for implementing the hybrid queries optimization. The interfaces and messages turn our optimization approach self-contained.

– Hypatiaaccepts the hybrid query evaluation requests. HybridQP validates the expression according to the information provided by the APIDirectory. QEPBuilder derives the optimization objective from the SLA contract and requests the hybrid query optimization to HyQoZ. The resulting query workflow is executed by the QWExecutor.
– HyQoZaccepts hybrid query optimization requests and looks for the satisfaction of the optimization objectives derived from the SLA contracts. HyQoZis composed by a series of components that implement the optimization stages.

Internally, HyQoZis composed by a series of orthogonal components that together perform the optimization. Components exchange self-descriptive messages carrying the required information for articulating the optimization.

ISSN 2395-8618

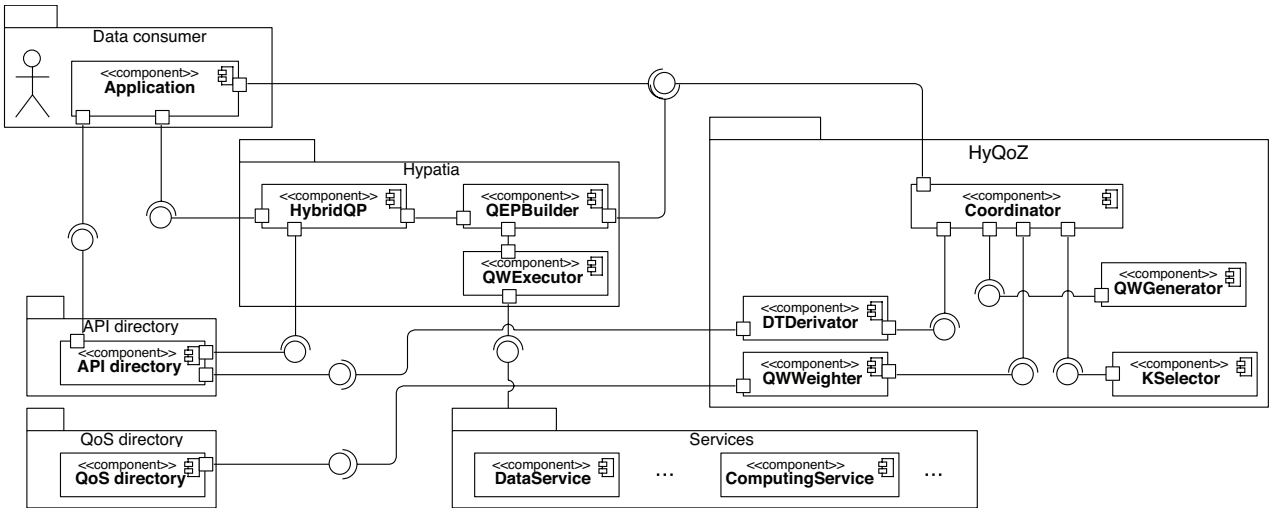


Fig. 5. Hybrid query processing components

B. Validation

We implemented two test scenarios and their corresponding data services to validate our approach. The first one, mainly a demonstration scenario, is the location-based application. In order to implement the Friend Finder scenario described in the Introduction we developed a test dataset using GPS tracks obtained from everytrail.com. Concretely, we downloaded 58 tracks corresponding to travel (either by walking, cycling, or driving) within the city of Paris. We converted the data from GPX to JSON and integrated it to our stream server to create the location service. For the profile and interests services we created a MySQL database accessible via JAX-WS Web Services running on Tomcat. The profile data is artificial and the interests were assigned and scored randomly using the most popular tags used in Flickr and Amazon. For the nearest-neighbor (NN) points of interest we converted a KML file[7] containing the major tourist destinations in Paris into JSON, this data is employed by the corresponding NN service in conjunction with the R-tree spatial indexation service. Finally, we implemented an interface based on Google Maps that enables to visualize the query result, which is presented in Figure 6.

The second scenario was developed to measure the efficiency of our current implementation in a more precise manner; it is based on the NEXMark benchmark[8]. Our main goal was to measure the overhead of using services, so we measured the total latency (i.e. the total time required to process a tuple present in the result) for a set of six queries (see Table I); first for our service-based system and then for an equivalent system that had at its disposal the same functionality offered by the computation services, but supported by objects inside the same Java Virtual Machine.

[7]Keyhole Markup Language https://developers.google.com/kml/documentation/

[8]http://datalab.cs.pdx.edu/niagara/NEXMark/

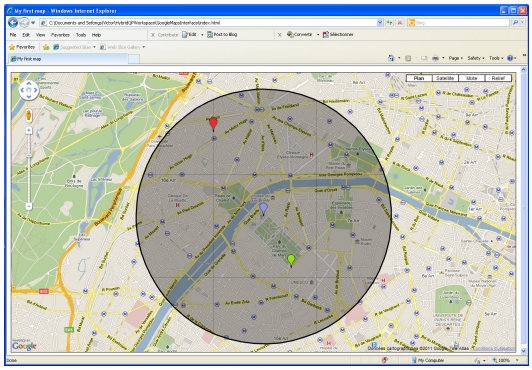


Fig. 6. Friend Finder visualization GUI

NEXMark proposes an auctions scenario consisting of three stream data services, **`person`**, **`auction`** and, **`bid`**, that export the following interfaces:

`person:`$\langle$`person_id`^{f}`, name`^{f}`, phone`^{f}`, email`^{f}`, income`$^f\rangle$

`auction:`$\langle$`open_auction_id`^{f}`, seller_person`^{f}`, category`^{f}`, quantity`$^f\rangle$

`bid:`$\langle$`person_ref`^{f}`, open_auction_id`^{f}`, bid`$^f\rangle$

Auctions work as follows. People can propose and bid for products. Auctions and bids are produced continuously. Table I shows the six queries that we evaluated in our experiment; they are stated in our HSQL language and for each we provide the associated query workflow and equivalent operator expression that implements them (generated by HYPATIA). Queries $\mathbf{Q}_1$ – $\mathbf{Q}_2$ mainly exploit temporal filtering using window operators, filtering and correlation with and/split-join like control flows. $\mathbf{Q}_3$ involves grouping and aggregation functions. $\mathbf{Q}_4$ adds a service call to a sequence of data processing activities with filtering and projection operations. Finally, $\mathbf{Q}_5$ – $\mathbf{Q}_6$ address several correlations organized in and/split-join control flows.

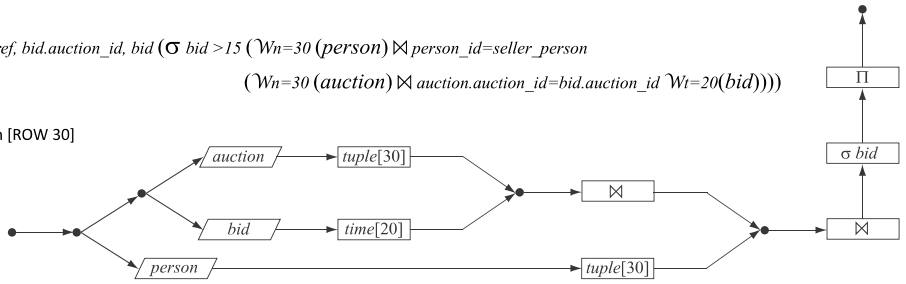
TABLE I
NEXMARK QUERIES

For the last 30 persons and 30 products offered, retrieve the bids of the last 20 seconds greater than 15 euros

1)

```
SELECT bids.person_ref, bid.auction_id, bid.bid
FROM bid [RANGE 20], auction [ROW 30], person [ROW 30]
WHERE bid.auction_id = auction.auction_id AND
auction.seller_person = person.person_id
AND bid.bid > 15;
```

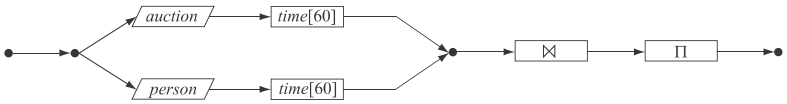
π *person_ref, bid.auction_id, bid* (σ *bid >15* ($\mathcal{W}$*n=30* (*person*) $\bowtie$ *person_id=seller_person*
($\mathcal{W}$*n=30* (*auction*) $\bowtie$ *auction.auction_id=bid.auction_id* $\mathcal{W}$*t=20*(*bid*))))



For the persons joining and the products offered during the last minute, generate the name and email of the person along with the id of the product he/she offers

2)

```
SELECT P.name, P.email, A.auction_id
FROM auction [RANGE 60] as A, person [RANGE 60] as P
WHERE A.seller_person = P.person_id;
```

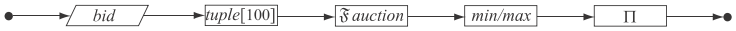


π *name, email, auction_id* ($\mathcal{W}$*t=60* (*person*) $\bowtie$ *person_id=seller_person* $\mathcal{W}$*t=60* (*auction*))

For the last 100 bids, find the maximum and minimum bid for each product

3)

```
SELECT bid.person_ref, bid.auction_id,
bid.bid, max(bid.bid), min(bid.bid)
FROM bid [ROWS 100]
GROUP BY bid.auction_id;
```

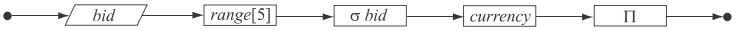


π *person_ref, auction_id, bid, max(bid), min(bid)* ($\mathfrak{F}$ *auction_id* ($\mathcal{W}$*t=60* (*auction*)))

Among the bids made in the last 5 seconds, find those whose amount is between 50 and 100 euros and their dollar equivalent

4)

```
SELECT bid.person_ref, bid.auction_id, bid.bid,
currency('EUR', 'USD', bid.bid)
FROM bid [RANGE 5]
WHERE bid.bid > 50.0 and bid.bid < 100.0;
```



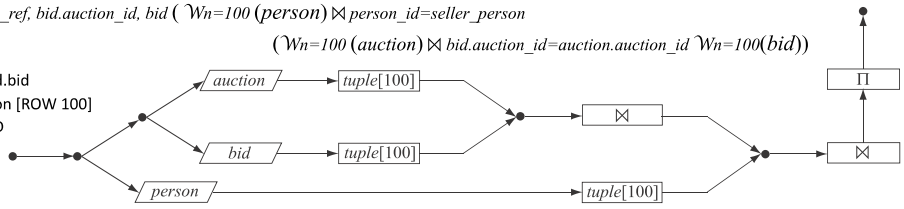
π *person_ref, auction_id, bid, currency('EUR', 'USD', bid)* (σ *bid >50 and bid <100* ($\mathcal{W}$*t=5* (*bid*)))

For the last 100 persons, products and bids; give the id of the seller person, the id of the product, and the amount of the bid

5)

```
SELECT bid.person_ref, bid.open_auction_id, bid.bid
FROM bid [ROW 100], auction [ROW 100], person [ROW 100]
WHERE bid.auction_id = auction.auction_id AND
auction.seller_person = person.person_id;
```

person_ref, bid.auction_id, bid ($\mathcal{W}$*n=100* (*person*) $\bowtie$ *person_id=seller_person*
($\mathcal{W}$*n=100* (*auction*) $\bowtie$ *bid.auction_id=auction.auction_id* $\mathcal{W}$*n=100*(*bid*))

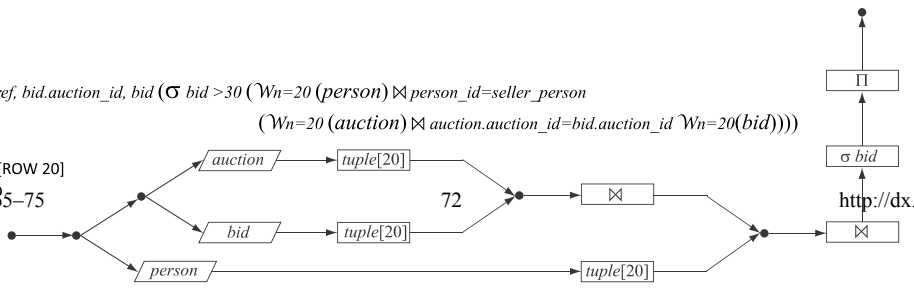


For the last 20 persons, products and bids; give the id of the seller person, the id of the product, and the amount of the bid, whenever that amount is greater than 30

6)

```
SELECT bid.person_ref, bid.auction_id, bid.bid
FROM bid [ROW 20], auction [ROW 20], person [ROW 20]
WHERE bid.auction_id = auction.auction_id AND
auction.seller_person = person.person_id
AND bid.bid > 30;
```

person_ref, bid.auction_id, bid (σ *bid >30* ($\mathcal{W}$*n=20* (*person*) $\bowtie$ *person_id=seller_person*
($\mathcal{W}$*n=20* (*auction*) $\bowtie$ *auction.auction_id=bid.auction_id* $\mathcal{W}$*n=20*(*bid*))))



 http://dx.doi.org/10.17562/PB-

ISSN 2395-8618

C. Experimental Results

For our experiments we used as a local machine a Dell D830 laptop with an Intel Core 2 Duo (2.10 GHz) processor and equipped with 2 GB of RAM. We also employed as a remote machine a Dell Desktop PC with a Pentium 4 (1.8 GHz) processor and 1 GB of RAM. In both cases running JSE 1.6.0_17, the local machine under Windows XP SP3 and the remote under Windows Server 2008.

As said before, to validate our approach we established a testbed of six queries based on our adaptation of the NEXMark benchmark. These queries include operators such as time and tuple based windows as well as joins. We measured tuple latency, i.e. the time elapsed from the arrival of a tuple to the instant it becomes part of the result, for three different settings. The first setting corresponds to a query processor using the same functionality of our computation services, but as plain java objects in the same virtual machine. In the second we used our computation services, which are based on the JAX-WS reference implementation, by making them run on a Tomcat container in the same machine as the query processor. For the third setting we ran the Tomcat container with the computation services on a different machine connected via intranet to the machine running the query processor.

The results are shown in Figure 7, and from them we derive two main conclusions. First, the use of services instead of shared memory resulted in about twice the latency. Second, the main overhead is due to the middleware and not to the network connection, since the results for the local Tomcat container and the remote Tomcat container are very similar. We believe that in this case the network costs are balanced-out by resource contingency on the query processor machine, when that machine also runs the container. We consider the overhead to be important but not invalidating for our approach, especially since in some cases we may be obliged to use services to acquire the required functionality.

From our experimental validation we learned that it is possible to implement query evaluation entirely relying on services without necessarily using a full-fledged DBMS or a DSMS (Data Stream Management Systems). Thereby, hybrid queries that retrieve on demand and stream data are processed by the same evaluator using well adapted operators according to their characteristics given our composition approach. The approach can seem costly because of the absence of a single DBMS, the use of a message based approach for implementing the workflow execution, and because there is no extensive optimization in the current version of HYPATIA. Now that we have a successful implementation of our approach, we can address performance issues further in order to reduce cost and overhead.

For validating HYQOZ, we developed the testbed that

1) generates synthetic hybrid queries,
2) generates the search space of query workflows following a data flow or control flow, and
3) estimates either the cost by means of a simulation using synthetic data statistics.

We used precision and recall measures to determine the proportion of interesting query workflows that are provided by the optimizer. The precision and recall are around 70% and 60% respectively.

V. RELATED WORK

In dynamic environments, query processing has to be continuously executed as services collect periodically new information (e.g., traffic service providing information at given intervals about the current state of the road) and the execution context may change (e.g., variability in the connection). Query processing should take into account not only new data events but also data providers (services) which may change from one location to another.

Existing techniques for handling continuous spatio-temporal queries in location-aware environments (e.g., see [6]–[11]) focus on developing specific high-level algorithms that use traditional database servers [12]. Most existing query processing techniques focus on solving special cases of continuous spatio-temporal queries: some like [8], [10], [11], [13] are valid only for moving queries on stationary objects, others like [14], [15] (Carney et al. 2002) are valid only for stationary range queries. A challenging perspective is to provide a complete approach that integrates flexibility into the existing continuous, stream, snapshot, spatio-temporal queries for accessing data in pervasive environments.

The emergence of data services has introduced a new interest in dealing with these “new” providers for expressing and evaluating queries. Languages as Pig and LinQ combine declarative expressions with imperative ones for programming queries, where data can be provided by services. In general, query rewriting, optimization and execution are the evaluation phases that need to be revisited when data are provided by services and they participate in queries that are executed in dynamic environments. Query rewriting must take into consideration the data service interfaces, since some data may need to be supplied to these in order to retrieve the rest of the data. The existence of a large number of heterogeneous data services may also necessitate the use of data integration techniques. In addition, new types of queries will require the definition of new query operators.

Traditional query optimization techniques are not applicable in this new setting, since the statistics used in cost models are generally not available. Furthermore, resources will be dynamically allocated via computation services, rather than being fixed and easy to monitor and control. Finally, query execution must also be reconsidered. First, the means to access the data is via services rather than scanning or employing index structures. Second, to process the data we depend on computation services, instead of a rigid DBMS.

The work [16] proposes a service coordination approach, where coordinations can optimized by ordering the service

ISSN 2395-8618

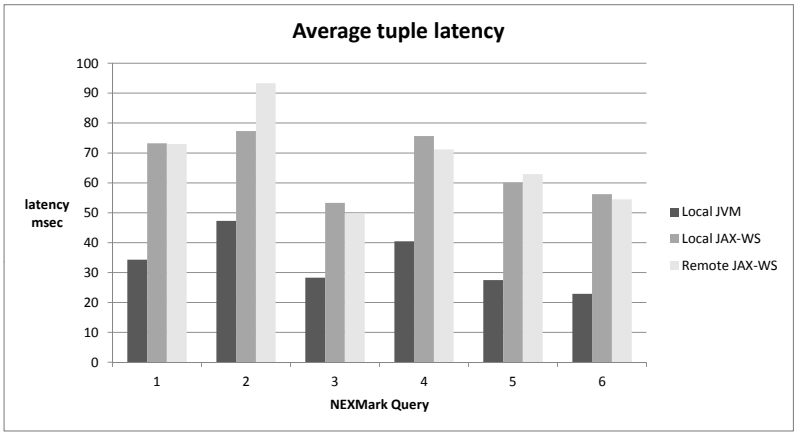


Fig. 7. Tuple latency for a services-based vs. a single Java application query processor

calls in a pipelined fashion and by tuning the data size. The control over data size (i.e., data chunks) and selectivity statistics are key assumptions adopted by the approach. Another aspect to consider during the optimization is the selection of services, which can have an impact on the service coordination cost. The authors of [3], [17] optimize service coordinations by proposing a strategy to select services according to multidimensional cost. Service selection is done by solving a multi objective assignment problem given a set of abstract services defined by the coordination. Services implementing the coordination can change but the control flow of the coordination remains the same.

The emergence of the map-reduce model, has introduced again parallelization techniques. Queries expressed in languages such as Pig[9], and SCOPE [18] can be translated into map/reduce [19] workflows that can be optimized. The optimization is done by intra-operator parallelization of map/reduce tasks. The work [20] applies safe transformations to workflows for factorizing the map/reduce functions, partitioning of data, and reconfiguring functions. Transformations hold preconditions and postconditions associated to the functions in order to keep the data flow consistency. The functional programming model PACT [21] extends the map/reduce model to add expressiveness that are black boxes within a workflow. In [22] the black boxes are analyzed at build-time to get properties and to apply conservative reorderings to enhance the run-time cost. The map/reduce workflows satisfy the need to process large-scale data efficiently w.r.t. execution time. Although we do not address query optimization under such context in the present work, we provide a discussion of its related issues and possible solutions in [23].

VI. Conclusion and Future Work

This paper presented our approach for optimizing service coordinations implementing queries over data produced by data services either on-demand or continuously. Such queries are implemented by query workflows that coordinate data and computing services. The execution of query workflows has to respect Service Level Agreement contracts that define an optimization objective described by a vector of weighted cost attributes such as the price, the time, the energy. The weights define the preferences among the cost attributes for enabling the comparison among query workflows. Our approach for generating the search space of query workflows that can optimize service coordinations, the cost estimation, and the solution space are oriented to satisfy SLA contracts.

[9]http://pig.apache.org

Acknowledgment

The authors would like to thank Víctor Cuevas Vicenttin and Carlos Manuel López Enríquez who were at the origin of the work presented here. We also thank the ANR ARPEGE project OPTIMACS that financed part of the work.

References

[1] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," *The VLDB Journal*, vol. 15, no. 2, pp. 121–142, 2006.

[2] U. Jaeger, "SMILE — a framework for lossless situation detection," in *In Proc. Int'l. Workshop on Inf. Technologies and Sys*, 1995, pp. 110–119.

[3] H. Wada, P. Champrasert, J. Suzuki, and K. Oba, "Multiobjective optimization of SLA-aware service composition," *2008 IEEE Congress on Services - Part I*, pp. 368–375, Jul. 2008.

[4] R. Fagin, A. Lotem, and M. Naor, "Optimal aggregation algorithms for middleware," *Journal of Computer and System Sciences*, vol. 66, no. 4, pp. 614–656, Jun. 2003.

[5] L. Martínez, C. Collet, C. Bobineau, and E. Dublé, "The QOL approach for optimizing distributed queries without complete knowledge," in *IDEAS*, 2012, pp. 91–99.

[6] R. Benetis, S. Jensen, G. Kariauskas, and S. Saltenis, "Nearest and reverse nearest neighbor queries for moving objects," *The VLDB Journal*, vol. 15, no. 3, pp. 229–249, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1007/s00778-005-0166-4

[7] I. Lazaridis, K. Porkaew, and S. Mehrotra, "Dynamic queries over mobile objects," in *Proceedings of the 8th International Conference on Extending Database Technology: Advances in Database Technology*, ser. EDBT'02. London, UK, UK: Springer-Verlag, 2002, pp. 269–286. [Online]. Available: http://dl.{ACM}.org/citation.cfm?id=645340.650229

ISSN 2395-8618

[8] Z. Song and N. Roussopoulos, "K-Nearest neighbor search for moving query point," in *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, ser. SSTD'01. London, UK, UK: Springer-Verlag, 2001, pp. 79–96. [Online]. Available: http://dl.{ACM}.org/citation.cfm?id=647227.719093

[9] Y. Tao, X. Xiao, and R. Cheng, "Range search on multidimensional uncertain data," *ACM Trans. Database Syst.*, vol. 32, no. 3, Aug. 2007. [Online]. Available: http://doi.{ACM}.org/10.1145/1272743.1272745

[10] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, ser. SIGMOD'03. New York, NY, USA: ACM, 2003, pp. 443–454. [Online]. Available: http://doi.{ACM}.org/10.1145/872757.872812

[11] B. Zheng and D. L. Lee, "Semantic caching in location-dependent query processing," in *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases*, ser. SSTD'01. London, UK, UK: Springer-Verlag, 2001, pp. 97–116. [Online]. Available: http://dl.{ACM}.org/citation.cfm?id=647227.719097

[12] M. F. Mokbel, X. Xiong, and W. G. Aref, "SINA: Scalable incremental processing of continuous queries in spatio-temporal databases," in *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, G. Weikum, A. C. Koenig, and S. Dessloch, Eds. ACM, 2004, pp. 623–634.

[13] O. Wolfson, A. P. Sistla, S. Chamberlain, and Y. Yesha, "Updating and querying databases that track mobile units," *Distrib. Parallel Databases*, vol. 7, no. 3, pp. 257–387, Jul. 1999. [Online]. Available: http://dx.doi.org/10.1023/A:1008782710752

[14] M. Hadjieleftheriou, G. Kollios, D. Gunopulos, and V. J. Tsotras, "On-line discovery of dense areas in spatio-temporal databases," in *SSTD*, 2003, pp. 306–324.

[15] S. Prabhakar, Y. Xia, D. V. Kalashnikov, W. G. Aref, and S. E. Hambrusch, "Query indexing and velocity constrained indexing: Scalable techniques for continuous queries on moving objects," *IEEE Trans. Comput.*, vol. 51, no. 10, pp. 1124–1140, Oct. 2002. [Online]. Available: http://dx.doi.org/10.1109/TC.2002.1039840

[16] U. Srivastava, K. Munagala, J. Widom, and R. Motwani, "Query optimization over web services," *VLDB'06, Proceedings of the 32nd International Conference on Very Large Data Bases*, 2006.

[17] D. Claro Barreiro, P. Albers, and J.-k. Hao, "Selecting web services for optimal composition," in *International Conference on Web Services (ICWS05)*, 2005.

[18] R. Chaiken, B. Jenkins, and Larson, "Scope: easy and efficient parallel processing of massive data sets," *VLDB*, 2008.

[19] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 1–13, 2008.

[20] H. Lim, "Stubby: A transformation-based optimizer for MapReduce workflows," *VLDB*, vol. 5, no. 11, pp. 1196–1207, 2012.

[21] D. Battré, S. Ewen, F. Hueske, and O. Kao, "Nephele/Pacts: A programming model and execution framework for web-scale analytical processing," *Proceedings of the 1st ldots*, 2010.

[22] F. Hueske, M. Peters, M. J. Sax, and A. Rheinl, "Opening the black boxes in data flow optimization," *VLDB*, vol. 5, no. 11, pp. 1256–1267, 2012.

[23] V. Cuevas-Vicenttin, G. Vargas-Solar, C. Collet, and P. Bucciol, "Efficiently coordinating services for querying data in dynamic environments," *Mexican International Conference on Computer Science*, vol. 0, pp. 95–106, 2009.