The HYPATIA System for Processing Hybrid Queries *

Víctor Cuevas-Vicentín*, Christine Collet†, Genoveva Vargas-Solar*, and Noha Ibrahim*

*CNRS, Grenoble Institute of Technology, University Joseph Fourier Grenoble Informatics laboratory (LIG),
BP 72, 38402, Saint Martin d’Hères Cedex, France
†French Mexican Laboratory of Informatics and Automatic Control Exhacienda Sta. Catarina Mártr s/n 72820
San Andrés Cholula, Puebla, México

{Firstname.Lastname}@imag.fr

ABSTRACT
The emergence of ambient computing introduces wireless and portable technologies that democratize access to information and data through services and thereby opens new research challenges for data querying. The most popular method to access data within these novel dynamic execution environments in a convenient and efficient way is still to consider declarative queries. Such queries, so-called hybrid queries may involve streaming and on demand data originated from services, possibly with temporal and mobile properties. For evaluating these queries we propose the HYPATIA system that tackles two main issues (i) using service coordination for building query plans and (ii) efficient and flexible evaluation capabilities.

1. INTRODUCTION
The advent of mobile computing and communication technologies lays a foundation for enabling timely and inexpensive access to information. We view declarative queries as the most convenient way of expressing information needs in an ambient computing environment. Such queries involve streaming and on demand data originated from services, which may have temporal and mobile properties; due to these characteristics we refer to them as hybrid queries.

Traditional techniques in conjunction with techniques used for evaluating queries over data streams could be a solution for processing hybrid queries. Adopting such an approach, however, presents two major drawbacks; both of which are associated with traditional DBMSs, but that are particularly disadvantageous in highly dynamic environments. First, a query management system based on such an approach will be expensive to develop and maintain. The addition of new capabilities in particular, would require that highly qualified system programmers develop the extensions in the chosen programming language, after which it is necessary to recompile and redeploy the system. Second, the system would require a heavy dedicated infrastructure, making the system difficult to scale and yielding it confined to a particular platform and location.

To our knowledge none of existing querying techniques tackle at the same time classic, mobile and continuous queries by composing services that are (push and pull as well as static and nomad) data services/providers and computation services providing computing functionalities such as query operators in particular. The main contributions of the HYPATIA system we proposed for evaluating hybrid queries are: (i) the use of a service-based approach for building query plans. Activities of such coordinations correspond to service calls, where services wrap either classical DBMSs, search engines, or specific services providing computing functions. (ii) the way hybrid query coordinations are evaluated. Indeed, our flexible query processing approach is based on a core data processing infrastructure providing adaptable service coordination mechanisms and data processing services (i.e., query operators). This paper first introduces the HYPATIA approach through a scenario. Then, section 3 describes how a query coordination is generated from a given declarative query and how it is evaluated. It also discusses the benefits that result from our approach, in particular regarding flexibility. Finally, section 4 describes the system itself showing the feasibility of this approach for the evaluation of several types of queries without relying on a prebuilt and to a large degree inflexible DBMS.

2. HYPATIA OVERVIEW
The goal of HYPATIA is to provide an all-encompassing approach that integrates continuous, stream, snapshot, spatio-temporal, and mobile queries for accessing data in ambient computing environments. For this purpose, it introduces the notion of hybrid queries and a novel approach for flexible query processing using service coordination.

2.1 Hybrid Queries
To illustrate hybrid queries consider the scenario depicted in Figure 1. Multiple users are in an urban area carrying
GPS-enabled mobile devices that periodically transmit their location; furthermore, they have agreed to share some of their personal information. A user in this scenario may want to find friends which are no more than 3 km away from him, their personal information. A user in this scenario may want to find friends which are over 21 years old and that are interested in art.

To answer this query three data services need to be accessed, which produce data in one of two ways: on-demand in response to a given request, or continuously as a data stream. In either case, the data service exposes an interface, composed of several operations and supported by standardized protocols. The JavaScript Object Notation \(^1\) is used to represent the data. Accordingly, objects are built from atomic values, nested tuples, and lists.

The users' location is made available by a stream data service with the (simplified) interface

\[
\text{subscribe}(\text{location:}\langle\text{nickname, coor}\rangle)
\]

consisting of a subscription operation that after invocation, will produce a stream of location tuples with a nickname that identifies the user and his/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

\[
\begin{align*}
\text{profile(nickname)} & \rightarrow \text{person:}\langle\text{age, gender, email}\rangle \\
\text{interests(nickname)} & \rightarrow \{\text{s_tag:}\langle\text{tag, score}\rangle\}
\end{align*}
\]

The first provides a single person tuple denoting a profile of the user, once given a request represented by his/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, etc.) and a numeric score indicating the corresponding degree of interest.

The query under discussion can be expressed in a language similar to CQL\(^1\) as follows

\[
\begin{align*}
\text{SELECT} & \ p.\text{nickname, p.age, p.gender, p.email} \\
\text{FROM} & \ \text{profile p, location l [range 10 min], interests i} \\
\text{WHERE} & \ p.\text{age} \geq 21 \ \text{AND l.nickname = p.nickname AND} \\
& \ i.\text{nickname = p.nickname AND contains(i, \ 'tag', \ 'art')} \\
& \ \text{AND distance(l.coor, mycoor) < 3}
\end{align*}
\]

We rely on a special function distance to evaluate the spatial predicate of the query, which receives the current location of the user issuing the query as mycoor. Another special function, contains, is used to determine if the value 'art' occurs in the s_tag list for the tag attribute. The location stream is bounded by a time-based window which will consider only the data received within the last 10 minutes.

---

\(^1\) JSON Model http://www.json.org/
3.2 Query evaluation

Evaluating a hybrid query thus depends first on finding the adequate (data and computation) services, second on their invocation and execution, and finally on their communication and interoperation. We briefly deal with these aspects next.

Service invocation and execution

As previously discussed, we have on demand and streaming data services. Data is gathered from on demand data services by invoking their data operations with the appropriate parameters, producing as a result an output tuple. This process is depicted in the figure below for the profile service of our example. Streaming data services are treated in an analogous manner, except that a single invocation is performed on their subscribe operation, after which they produce a data stream that is sent to the destination specified with the subscription parameters.

On the other hand, the computation services that emulate query operators can be simple or composite. Simple computation services are those whose execution is performed by a single service operation invocation. Figure 3 at the left illustrates such a service. The distance computation service relies on a geo-distance service, which provides the capability to calculate the geographical distance between two points. This is achieved by the invocation of its distance operation with the appropriate parameters.

In contrast, the execution of a composite computation service involves multiple operation invocations, possibly also from different services, as well as manipulation of local data. These tasks are organized in a service coordination specified as a workflow, following a model in which we add conditional and iteration constructs to the ASM formalism.

The local data items, which are only visible within the composite computation service during its execution, maintain relevant information to guide the evaluation of the coordination appropriately. We should remark that composite computation service coordinations are executed repeatedly, on the recently arriving input and based on the current state of the data items.

Figure 3 at the right gives an example of a composite computation service, in the form of an overview of a service evaluating the join operator based on the symmetric hash-join algorithm and two instances of a stateful hash-index service. Several interrelated operation invocations on both service instances, as well as reads and updates on local data items, are used to find the tuple matches that form part of the join result.

Service interoperation and communication

The computation services in a query coordination communicate via asynchronous input operations exposed by its executing environment. In order to take advantage of this mechanism, composite computation services must be designed in such a way that they access the input data sent to them by their preceding computation services. This data communication occurs when a service invokes the corresponding input operations of its successor, which in turn will store the input tuples as part of their state in order to process it accordingly. Figure 4 depicts this process.

3.3 Flexibility

Our query processing approach based on service coordination is geared towards flexibility, which is highly desirable in ambient computing environments where data providers can become unavailable. First, service coordination offers the capability to dynamically acquire resources by adopting a late binding approach where the best services available in the environment are bound at the query evaluation time. As evaluation may be continuous access to certain services that are subject to frequent disconnections is an important issue to handle. Our system enables the replacement of services whenever such failures arise. Generally speaking if some data services or computation services become unavailable, evaluating the query is still possible by discovering other services in order to access the same or similar data and functionality in the environment.

For instance, if the geo-distance service discussed previously becomes unavailable, we can rely on an alternative service. Concretely, a service offering an spatial index based on an R-Tree could be used instead. In this case we can determine the tuples that satisfy the spatial condition by creating an index over them. This form of flexibility also opens opportunities for optimization.

4. HYPATIA ARCHITECTURE

The architecture of the HYPATIA service-based hybrid query processor is presented in Figure 5, it was developed on the Java platform.

Queries in HYPATIA are entered via a GUI (presented in Figure 6) and specified in a query language similar to CQL[1], complemented with function calls. Once a query is provided to the system and the user indicates it should begin
execution, it is parsed and then its corresponding query coordination is generated by the query parser and the query coordination constructor components, respectively. The parser was developed using the ANTLR parser generator.

The GUI also enables the user to visualize the query coordination (at the middle left of Figure 6), which is facilitated by the use of the JGraph library. Data services are represented in yellow whereas computation services are represented in blue, both with their corresponding labels.

The evaluation of a given query is enabled by two main components that support the computation services corresponding to query operators. A scheduler determines which service is executed at a given time according to a predefined policy, while composite services coordinations are executed by the workflow engine that implements our workflow model, which extends our previous work[3].

The workflow corresponding to the coordination of a composite computation service is interpreted by the engine, also developed with the ANTLR parser generator. These computation service workflows can also be visualized through the GUI, as shown in the right of Figure 6.

We developed a set of computation services that are used to build hybrid query operators. These services run on a Tomcat container supported by the JAX-WS reference implementation, which enables to create stateful services. The operators currently implemented are join and bind-join, tuple and time based windows, selection, and projection. Additional services enable to evaluate predicates specified through special purpose functions as in our example query.

The data corresponding to data stream services is supplied by a subscription-based stream server relying on Web service standards, specially developed for that purpose and complemented with mechanisms to control and synchronize the streams. On demand data services, on the other hand, are implemented as conventional Web services exposed through Tomcat.

During the evaluation of a query data tuples flow from the data services to various computation services, as determined by the query coordination. Computation services communicate among each other via asynchronous queues implementing input operations. The end result is a data stream that denotes the tuples that are added and the tuples that are removed from the query answer, which is indicated by a special (positive or negative, respectively) numeric attribute on the tuples. The query result stream is presented in a textual form in the GUI (bottom of Figure 6).

We implemented two test scenarios and their corresponding data services to validate our approach. The first one is the location-based application introduced in Section 2.1, and which is the subject of our demonstration. The second scenario was developed to measure the efficiency of our current implementation in a more reliable manner; it is based on the NEXMark benchmark.

5. REFERENCES


