Coordination through Querying in the Youtopia System

[Demonstration paper]

Nitin Gupta, Lucja Kot, Gabriel Bender, Sudip Roy and Johannes Gehrke
Cornell University
Ithaca, NY 14853, USA
{niting, lucja, gbender, sudip, johannes}@cs.cornell.edu

Christoph Koch
EPFL
CH-1015 Lausanne, Switzerland
christoph.koch@epfl.ch

ABSTRACT

In a previous paper, we laid out the vision of declarative data-driven coordination (D3C) where users are provided with novel abstractions that enable them to communicate and coordinate through declarative specifications [3].

In this demo, we will show Youtopia, a novel database system which is our first attempt at implementing this vision. Youtopia provides coordination abstractions *within* the DBMS. Users submit queries that come with explicit coordination constraints to be met by other queries in the system. Such queries are evaluated together; the system ensures that their *joint* execution results in the satisfaction of all coordination constraints. That is, the queries *coordinate* their answers in the manner specified by the users.

We show how Youtopia and its abstractions simplify the implementation of a three-tier flight reservation application that allows users to coordinate travel arrangements with their friends.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Design

1. INTRODUCTION

In the long history of humankind (and animal kind, too) those who learned to collaborate and improvise most effectively have prevailed. — Charles Darwin.

The abstraction of isolation has long been one of the main functionalities provided by a database system. We have developed Youtopia, a system that — in addition to isolation through transactions — provides Web applications with abstractions that allow users to *coordinate* actions. Programs can *selectively give up isolation* in order to accomplish a larger task. In particular, our system allows users to register *entangled queries* that can only be answered in conjunction with other entangled queries posed by other users. The system evaluates sets of such queries jointly in order to ensure coordinated answers. We make this notion of coordination more concrete by introducing the example we showcase in our demo.

We will show a travel Web site that enables users to coordinate flight and hotel reservations. For example, suppose two friends, Jerry and Kramer, plan a vacation together. Each wants to book a seat on a flight from New York to Paris that satisfies certain date and price constraints. They also want to be on the same flight. They would like to coordinate their bookings while still performing them individually. The idea is to avoid delegating the booking of both seats to one person, or coordinating out-of-band to choose the flight and trying to make near-simultaneous bookings. Conceptually, the two booking queries should run "jointly" even though they were submitted individually and at different times. The users want to give up isolation in a principled way in order to achieve coordination. Youtopia supports this functionality, and it does so within the DBMS, freeing the developers of the travel website from the burden of implementing it in the middle tier.

Youtopia supports coordination through *entangled queries* [2]. These queries use special answer constraints as a means of coordination. The idea is that the answer to the query is returned through an answer relation that is shared among multiple queries in the system. An individual query can only be answered if the system-wide answer relation satisfies a postcondition specified with the query itself. This postcondition can, and usually will, pertain to answers of other queries in the system. For example, a user can specify an entangled query that states "please give me the number of a flight to Paris, but only if my friend's query about the same subject receives the same flight number". A query whose postcondition is not satisfied is not rejected but waits for an opportunity to retry.

We believe that Youtopia captures coordination at the ideal level of abstraction [3]. On one hand, entangled queries are simpler to use than lower-level coordination constructs offered by operating systems; on the other, they support many interesting use cases. They also add real power for new applications: In particular, the functionality of matching and jointly executing entangled queries is not supported by triggers, nested transactions [4], or sagas [1].

Focus of this demonstration. We will demonstrate Youtopia and its entangled query evaluation algorithm [2] using the travel Web application described above, showing how entangled queries allowed us to build this application with ease. We will also demonstrate how query matching and execution works inside Youtopia.

2. OUR SYSTEM

In this section, we briefly introduce entangled queries and present the Youtopia system. A more complete treatment is found in our associated technical paper [2].

2.1 Entangled Queries

An entangled query has the following syntax. It is essentially a SELECT statement extended with coordination constraints.

SIGMOD'11, June 12–16, 2011, Athens, Greece.
ACM 978-1-4503-0661-4/11/06.

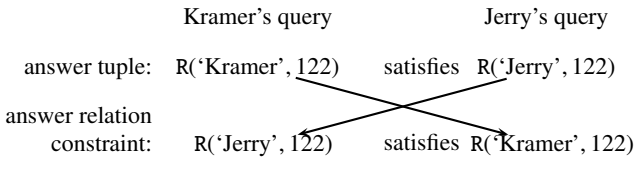
Flights

fno	dest
122	Paris
123	Paris
134	Paris
136	Rome

Airlines

fno	airlines
122	United
123	United
134	Lufthansa
136	Alitalia

(a)



(b)

Figure 1: (a) Flight database (b) Mutual constraint satisfaction

```
SELECT select_expr
INTO ANSWER tbl_name [, ANSWER tbl_name] ...
[WHERE where_answer_condition]
```

We illustrate the semantics of entangled queries with an example. Suppose Kramer, wants to travel to Paris on the same flight as Jerry. He can express his request with the following entangled query.

```
SELECT 'Kramer', fno INTO ANSWER Reservation
WHERE
fno IN (SELECT fno FROM Flights WHERE dest='Paris')
AND ('Jerry', fno) IN ANSWER Reservation
CHOOSE 1
```

`Reservation` is the name for the relation which will contain the answers to all the current queries in the system. The `SELECT` clause specifies Kramer's own expected answer, or, in other words, his contribution to the answer relation. This contribution, however, is conditional on two requirements, which are given in the `WHERE` clause. First, the flight number in question must correspond to a flight to Paris. Second, the answer table must also contain a tuple with the same flight number but `Jerry` as the traveler name. Clearly, if this query is evaluated by itself, the answer constraint cannot be satisfied. However, the query is not rejected, but rather gets registered in the system for possible later execution.

Suppose now Jerry submits a symmetric query which looks just like Kramer's, except that the strings `'Kramer'` and `'Jerry'` are swapped. Once the system has received both queries and recognized that they "match", it answers both of them simultaneously in a way that ensures a coordinated flight number choice. In general, there may be many different suitable flights, but Kramer and Jerry only want to make a booking on one of them. The semantics of our queries is such that each query only receives one answer tuple, as indicated by the `CHOOSE 1` clause. If the database is as shown in Figure 1 (a), the system nondeterministically chooses either flight 122 or 123 and returns appropriate answer tuples. Figure 1 (b) shows how the constraints between the queries are satisfied.

2.2 System Description

The system architecture of our demonstration is shown in Figure 2. Note that entangled queries are handled within the Youtopia system itself. Entangled queries are generated by the middle tier and submitted to Youtopia. The query compiler processes them and

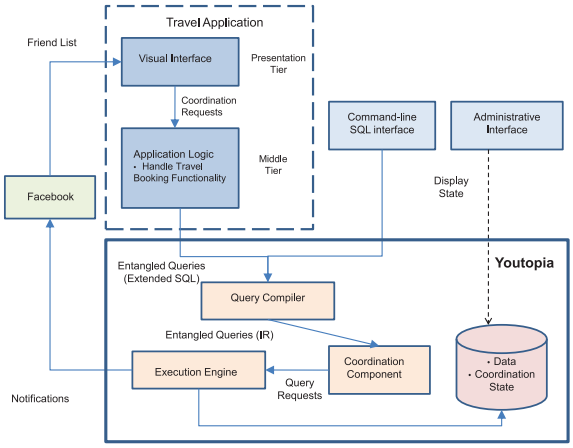


Figure 2: Demo Architecture

translates them to an intermediate representation inside Youtopia for processing by the coordination component. The coordination component runs whenever an entangled query arrives in the system. The coordination logic accesses regular database tables as well as other internal tables that store the list of pending queries [2]. The execution engine evaluates queries on the database as required by the coordination component, as well as executing any other queries and updates that may be necessary.

Our demonstration includes three applications that are built on top of Youtopia. The first application is the travel Web site that allows users to coordinate travel and hotel reservations with their Facebook friends. The application itself follows a standard three-tier architecture. The graphical frontend runs in a browser; it provides an interface to all the middle tier functionality. At the middle tier, we have implemented application logic to handle the standard functionality of a travel Web site such as searching for flights and hotels, selecting specific flights and hotels, and to create and coordinate new travel reservations based on the user's list of friends that is populated using the Facebook API. The application logic also contains an "account view" where a user can see pending or confirmed reservations.

Our second application is an SQL command line interface which allows SQL and entangled queries to be input directly to the system by the user. The third application is an administrative interface which allows us to show the internal state of the system and to visualize the state created by the matching algorithms.

3. DEMONSTRATION OUTLINE

We will demonstrate Youtopia's support for entangled queries using the travel plan coordination scenario. Our goal is to show that entangled queries are a clean, robust and powerful abstraction, and that they make it easy to implement coordination-driven applications which would be difficult to program using existing abstractions. We also demonstrate the scalability of our coordination algorithm by allowing our examples to be run on a loaded system, where a large number of entangled queries are trying to coordinate simultaneously.

3.1 Coordinating Travel with Youtopia

We will demonstrate the following scenarios.

Book a flight with a friend. The first example will show how

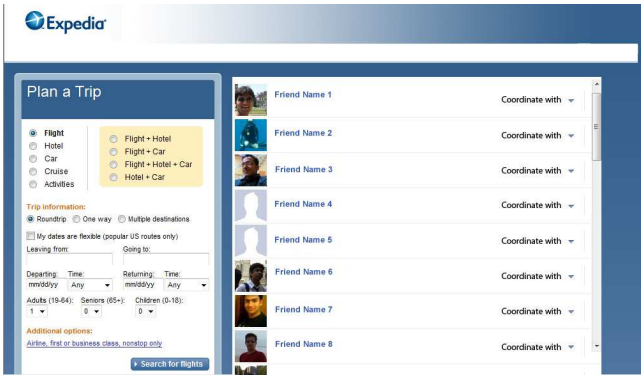


Figure 3: Choosing a Friend for Flight Coordination

Figure 4: Viewing Friends' Existing Flight Bookings

a user, Jerry, may use our system to coordinate flight plans with a Facebook friend, Kramer. He begins the process by logging in to Facebook so that Kramer's contact information can be imported.

Next, Jerry chooses Kramer from his list of friends; this is shown in Figure 3. He can now specify that he wants to fly in an adjacent seat to Kramer, or just that he wants to travel on the same flight. He submits his request, and the system translates it into an entangled query which is processed by Youtopia. Once the system is able to find a flight booking which allows him to coordinate with Kramer as he requested, it makes a flight reservation for him. Jerry is notified of the success of his request via a Facebook message.

Jerry may also follow an alternate path to achieve the same goal. He can browse for flights first and find out if any of his friends already have bookings on them. This is shown in Figure 4. If he decides he is able to choose a flight based on this information, he can go ahead and make his own booking directly through the system.

Book a flight and a hotel with a friend. In this scenario, Jerry is coordinating with Kramer on flight plans as before, but he also requests to stay in the same hotel. To accomplish this, Jerry begins by performing his flight coordination request as before. Next, he uses a very similar interface and mechanism to also specify a hotel coordination request with Kramer. Once he has completed both parts of his request, he submits an entangled query to the system that contains constraints on both the flight and the hotel. We will then show how Kramer can log into the system and submit a matching request. On submission of this second request, Youtopia jointly executes the entangled queries and obtains answers, which are then communicated to the users of our application via a Facebook message.

Multiple simultaneous bookings. Next, we will demonstrate how our system works when multiple users are concurrently trying to coordinate flight and hotel reservations together. This is a scenario analogous to the previous case, except that it involves multiple pairs of users trying to coordinate with each other.

Group flight booking. Users may wish to coordinate travel plans in groups larger than two. We demonstrate how a group of four friends can jointly specify that they wish to travel on the same flight to their joint destination. Each individual specifies their request through a process analogous to Jerry's workflow in the first example; however, instead of choosing just one friend, they select the entire group of friends that will be taking part in the trip. Again, the system processes their requests to make coordinated flight reservations for them.

Group flight and hotel booking. We show how a group of friends can coordinate not just on flights, but on hotels as well, just as in the two-person case.

Ad-hoc examples. Finally, we will demonstrate how other ad-hoc coordination scenarios can be created and supported in Youtopia. Arbitrary groups of users are able to coordinate on their travel and/or hotel reservation plans, in flexible ways. For example, it is possible to have a group of three friends, Jerry, Kramer and Elaine, where Jerry and Kramer coordinate on flight reservations only, whereas Kramer and Elaine coordinate on both flight and hotel reservations.

3.2 User Interfaces

In addition to the visual interface described above, we also have an administrative ("debugging") interface to Youtopia with a command line. The command line allows us to show how we can directly input SQL code into the system, specifying entangled queries on our travel database. Any of the above examples can be executed in this way, as well as any other arbitrary queries the user may care to specify.

Furthermore, the administrative interface has a special mode that enables visual inspection of the state of the system. This allows us to show to the audience facts about the system state such as the set of queries pending to be entangled and their representation in the system.

4. ACKNOWLEDGMENTS

This research has been supported by the NSF under Grants IIS-0534404 and IIS-0911036, by a Google Research Award, by NYSTAR under Agreement C050061, and by the iAd Project of the Research Council of Norway. Any opinions, findings, conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the sponsors.

5. REFERENCES

[1] H. Garcia-Molina and K. Salem. Sagas. *SIGMOD Rec.*, 16(3):249–259, 1987.

[2] N. Gupta, L. Kot, S. Roy, G. Bender, J. Gehrke, and C. Koch. Entangled queries: enabling declarative data-driven coordination. In *SIGMOD*, 2011.

[3] L. Kot, N. Gupta, S. Roy, J. Gehrke, and C. Koch. Beyond isolation: Research opportunities in declarative data-driven coordination. *SIGMOD Record*, 39(1):27–32, 2010.

[4] N. A. Lynch and M. Merritt. Introduction to the theory of nested transactions. *Theor. Comput. Sci.*, 62(1-2):123, 1988.