SIGMOD Officers, Committees, and Awardees

Chair	Vice-Chair	Secretary/Treasurer
Yannis Ioannidis	Christian S. Jensen	Alexandros Labrinidis
University of Athens	Department of Computer Science	Department of Computer Science
Department of Informatics	Aalborg University	University of Pittsburgh
Panepistimioupolis, Informatics Bldg	Selma Lagerlöfs Vej 300	Pittsburgh, PA 15260-9161
157 84 Ilissia, Athens	DK-9220 Aalborg Øst	
HELLAS	DENMARK	USA
+30 210 727 5224	+45 99 40 89 00	+1 412 624 8843
<yannis AT di.uoa.gr>	<csj AT cs.aau.dk >	<labrinid AT cs.pitt.edu>

[Last updated : Aug 1st, 2010]

SIGMOD Edgar F. Codd Innovations Award

For innovative and highly significant contributions of enduring value to the development, understanding, or use of database systems and databases. Until 2003, this award was known as the "SIGMOD Innovations Award." In 2004, SIGMOD, with the unanimous approval of ACM Council, decided to rename the award to honor Dr. E.F. (Ted) Codd (1923 - 2003) who invented the relational data model and was responsible for the significant development of the database field as a scientific discipline. Recipients of the award are the following:

Michael Stonebraker (1992)	Jim Gray (1993)	Philip Bernstein (1994)
David DeWitt (1995)	C. Mohan (1996)	David Maier (1997)
Serge Abiteboul (1998)	Hector Garcia-Molina (1999)	Rakesh Agrawal (2000)
Rudolf Bayer (2001)	Patricia Selinger (2002)	Don Chamberlin (2003)
Ronald Fagin (2004)	Michael Carey (2005)	Jeffrey D. Ullman (2006)
Jennifer Widom (2007)	Moshe Y. Vardi (2008)	Masaru Kitsuregawa (2009)
Umeshwar Dayal (2010)		

SIGMOD Contributions Award

For significant contributions to the field of database systems through research funding, education, and professional services. Recipients of the award are the following:

Maria Zemankova (1992)	Gio Wiederhold (1995)	Yahiko Kambayashi (1995)
Jeffrey Ullman (1996)	Avi Silberschatz (1997)	Won Kim (1998)
Raghu Ramakrishnan (1999)	Michael Carey (2000)	Laura Haas (2000)
Daniel Rosenkrantz (2001)	Richard Snodgrass (2002)	Michael Ley (2003)
Surajit Chaudhuri (2004)	Hongjun Lu (2005)	Tamer Özsu (2006)
Hans-Jörg Schek (2007)	Klaus R. Dittrich (2008)	Beng Chin Ooi (2009)
David Lomet (2010)		

SIGMOD Jim Gray Doctoral Dissertation Award

SIGMOD has established the annual SIGMOD Jim Gray Doctoral Dissertation Award to *recognize excellent research by doctoral candidates in the database field*. This award, which was previously known as the SIGMOD Doctoral Dissertation Award, was renamed in 2008 with the unanimous approval of ACM Council in honor of Dr. Jim Gray. Recipients of the award are the following:

- **2006** *Winner*: Gerome Miklau, University of Washington

Runners-up: Marcelo Arenas, Univ. of Toronto; Yanlei Diao, Univ. of California at Berkeley.

- **2007** *Winner*: Boon Thau Loo, University of California at Berkeley

Honorable Mentions: Xifeng Yan, UIUC; Martin Theobald, Saarland University

- **2008** *Winner*: Ariel Fuxman, University of Toronto

Honorable Mentions: Cong Yu, University of Michigan; Nilesh Dalvi, University of Washington.

- **2009** *Winner*: Daniel Abadi, MIT

Honorable Mentions: Bee-Chung Chen, University of Wisconsin at Madison; Ashwin Machanavajjhala, Cornell University.

- **2010** *Winner:* Christopher Ré (advisor: Dan Suciu), University of Washington

Honorable Mentions: Soumyadeb Mitra (advisor: Marianne Winslett), University of Illinois, Urbana-Champaign; Fabian Suchanek (advisor: Gerhard Weikum), Max-Planck Institute for Informatics

A complete listing of all SIGMOD Awards is available at: **http://www.sigmod.org/awards/**

[Last updated : Aug 1st, 2010]

Editor's Notes

Welcome to the March 2010 issue of the ACM SIGMOD Record. It is a pleasure and an honor to write my first Editor's Notes for Record.

We begin the issue with a regular article by Hellerstein. The article is a companion to his keynote talk at the ACM PODS conference this year. The article is based on the previous works carried at Berkeley on distributed and parallel processing, and in particular applied to the networking domain. It addresses the challenges and opportunities of declarative database languages, and in particular Datalog, for parallelizing computations. While the content has first been presented in the theory-oriented setting of PODS, the article balances theoretical aspects with many practical applications and implementation issues. The importance taken by parallel and distributed processing in today's CS industry makes this article very timely.

The second regular article is by Barbieri, Braga, Ceri, Della Valle and Grossniklaus. It presents C-SPARQL, an extension to the SPARQL query language for querying streams of RDF data, as well as applications and future research opportunities. C-SPARQL allows querying RDF data both from static repositories and from streams, enabling continuous reasoning in contexts such as sensor networks or social semantic data.

The last regular article of the issue proposes declarative data-driven coordination (D3C) and is authored by Kot, Gupta, Roy, Gehrke and Koch. The work is motivated by the need to be able to specify, in a high-level fashion, complex yet flexible coordination mechanisms, such as "I want to book a seat in the same flight as my friend" and have such specifications compiled in lower-level transactions to be executed by a DBMS.

Two articles appear in the **Surveys Column** (edited by Cesar Galindo-Legaria). First, Khare, An and Song analyze and compare for the first time techniques used to automatically understand Web interfaces. Extracting data from the "Deep Web" requires the capacity to automatically fill in forms in order to get content, and understanding the forms is crucial in order to provide appropriate input. The authors set a classification framework to analyze works in this area, and illustrate it by analyzing 10 existing approaches. Second, Drosou and Pitoura consider diversification of answers in search and recommender systems. Diversification here is considered from the angles of content, novelty and coverage. The authors review definitions of diversity and concrete algorithms for achieving it.

The **Systems and Prototypes Column** (edited by Magdalena Balazinska) features one article by Liu, Mihaylov, Bao, Jacob, Ives, Loo and Guha. The article describes the SmartCIS environment for integrating physical and virtual environments, and follows the demo given by the authors at SIGMOD 2009. The SmartCIS prototype is based on the Aspen project, which extends the data integration formalisms to a setting of distributed streams.

The issue also features two workshop reports. The first is authored by Benedikt, Florescu, Gardner, Guerrini, Mesiti and Waller and summarizes the workshop on *Updates for XML* which was held in conjunction with the EDBT 2010 conference in Lausanne. The topics addressed in the workshop considered updates on XML and other data formats, such as RDF and probabilistic data. The second reports on the *Cloud Data Management Workshop* held next to the ACM CIKM 2009 conference in Hong Kong. The report witnesses of the interest raised by cloud computing within the data management community and concludes that (at the time of the workshop) much remained to be done in order to realize that potential.

Our issue closes with the announcements of DMSN, the Seventh International Workshop on Data Management for Sensor Networks, and XLDB, the 4th Extremely Large Database Conference, to be held in Menlo Park in October 2010.

This issue appears with some delay that we hope to absorb until the end of the year, by publishing the next three issues at short intervals in order to get closer to the normal publication rhythm. The SIGMOD Executive Committee continues consulting over the new editorial policy of the SIGMOD Record; we expect to publicly announce it in the fall.

Ioana Manolescu
August 2010

Past SIGMOD Record Editors:

Harrison R. Morse (1969)
Daniel O'Connell (1971 – 1973)
Randall Rustin (1975)
Thomas J. Cook (1981 – 1983)
Jon D. Clark (1984 – 1985)
Margaret H. Dunham (1986 – 1988)
Arie Segev (1989 – 1995)
Jennifer Widom (1995 – 1996)
Michael Franklin (1996 – 2000)
Ling Liu (2000 – 2004)
Mario Nascimento (2005 – 2007)
Alexandros Labrinidis (2007 – 2009)

The Declarative Imperative

Experiences and Conjectures in Distributed Logic

Joseph M. Hellerstein
University of California, Berkeley
hellerstein@cs.berkeley.edu

ABSTRACT

The rise of multicore processors and cloud computing is putting enormous pressure on the software community to find solutions to the difficulty of parallel and distributed programming. At the same time, there is more—and more varied—interest in data-centric programming languages than at any time in computing history, in part because these languages parallelize naturally. This juxtaposition raises the possibility that the theory of declarative database query languages can provide a foundation for the next generation of parallel and distributed programming languages.

In this paper I reflect on my group's experience over seven years using Datalog extensions to build networking protocols and distributed systems. Based on that experience, I present a number of theoretical conjectures that may both interest the database community, and clarify important practical issues in distributed computing. Most importantly, I make a case for database researchers to take a leadership role in addressing the impending programming crisis.

This is an extended version of an invited lecture at the ACM PODS 2010 conference [32].

1. INTRODUCTION

This year marks the forty-fifth anniversary of Gordon Moore's paper laying down the Law: exponential growth in the density of transistors on a chip. Of course Moore's Law has served more loosely to predict the doubling of computing efficiency every eighteen months. This year is a watershed: by the loose accounting, computers should be 1 Billion times faster than they were when Moore's paper appeared in 1965.

Technology forecasters appear cautiously optimistic that Moore's Law will hold steady over the coming decade, in its strict interpretation. But they also predict a future in which continued exponentiation in hardware performance will only be available via parallelism. Given the difficulty of parallel programming, this prediction has led to an unusually gloomy outlook for computing in the coming years.

At the same time that these storm clouds have been brewing, there has been a budding resurgence of interest across the software disciplines in data-centric computation, including declarative programming and Datalog. There is more—and more varied—applied activity in these areas than at any point in memory.

The juxtaposition of these trends presents stark alternatives. Will the forecasts of doom and gloom materialize in a storm that drowns out progress in computing? Or is this the long-delayed catharsis that will wash away today's thicket of imperative languages, preparing the ground for a more fertile declarative future? And what role might the database community play in shaping this future, having sowed the seeds of Datalog over the last quarter century?

Before addressing these issues directly, a few more words about both crisis and opportunity are in order.

1.1 Urgency: Parallelism

> *I would be panicked if I were in industry.*
> — John Hennessy, President, Stanford University [35]

The need for parallelism is visible at micro and macro scales. In microprocessor development, the connection between the "strict" and "loose" definitions of Moore's Law has been severed: while transistor density is continuing to grow exponentially, it is no longer improving processor speeds. Instead, chip manufacturers are packing increasing numbers of processor cores onto each chip, in reaction to challenges of power consumption and heat dissipation. Hence Moore's Law no longer predicts the clock speed of a chip, but rather its offered degree of parallelism. And as a result, traditional sequential programs will get no faster over time. For the first time since Moore's paper was published, the hardware community is at the mercy of software: only programmers can deliver the benefits of the Law to the people.

At the same time, Cloud Computing promises to commoditize access to large compute clusters: it is now within the budget of individual developers to rent massive resources in the worlds' largest computing centers. But again, this computing potential will go untapped unless those developers can write programs that harness parallelism, while managing the heterogeneity and component failures endemic to very large clusters of distributed computers.

Unfortunately, parallel and distributed programming today is challenging even for the best programmers, and unworkable for the majority. In his Turing lecture, Jim Gray pointed to discouraging trends in the cost of software development, and presented *Automatic Programming* as the twelfth of his dozen grand challenges for computing [26]: develop methods to build software with orders of magnitude less code and effort. As presented in the Turing lecture, Gray's challenge concerned sequential programming. The urgency and difficulty of his twelfth challenge has grown markedly with the technology

trends toward parallelism. Hence the spreading cloud of doom and gloom.

1.2 Resurgency: Springtime for Datalog

In the spring time, the only pretty ring time
When birds do sing, hey ding a ding, ding;
Sweet lovers love the spring.
— Shakespeare

With these storm clouds on the horizon, it should be a matter of some cheer for database theoreticians that Datalog variants, like crocuses in the snow, have recently been seen cropping up outside the walled garden of PODS. Datalog and related languages have been proposed for use in a wide range of practical settings including security and privacy protocols [37, 20, 79], program analysis [43, 73, 28], natural language processing [21, 70], probabilistic inference [8, 71], modular robotics [7], multiplayer games [74], telecom diagnosis [1], networking [46] and distributed systems [3]. The renewed interest appears not to be the result of a coordinated effort, but rather (to hybridize metaphors) a grassroots phenomenon arising independently in different communities within computer science.

Over the past few years, my group has nurtured a patch of this activity in the unlikely ground of Berkeley's systems projects, with a focus on inherently parallel tasks in networking and distributed systems. The effort has been quite fruitful: we have demonstrated full-featured Datalog-style implementations of distributed systems that are orders of magnitude more compact than popular imperatively implemented systems, with competitive performance and significantly accelerated software evolution [46, 3]. Evidence is mounting that Datalog can serve as the rootstock of a much simpler family of languages for programming serious parallel and distributed software. Encouraged by these results, we are cultivating a new language in this style for Cloud Computing, which we call *Bloom*[1].

1.3 Synergy: The Long-Awaited Question

It shall be:
when I becloud the earth with clouds,
and in the clouds my bow is seen,
I will call to mind my covenant
that is between me and you and all living beings—
all flesh: never again shall the waters become a
Deluge, to bring all flesh to ruin!
– *Genesis*, 8:14-15 [24]

Though Gray speaks only vaguely about "non-procedural" languages in his Turing lecture, it is hard to imagine he did not have in mind the success of SQL over COBOL as one model for progress[2] And parallelism has proved quite tractable in the SQL context. Recently, David Patterson wrote soberly of the "Dead Computer Society" of parallel hardware vendors in the 1980's [35], but notably omitted the survivor from that era: parallel database pioneer Teradata. It happens that the relational algebra parallelizes very naturally over large datasets, and SQL programmers benefit without modifications to their code. This point has been rediscovered and amplified via the recent enthusiasm for MapReduce programming and "Big Data," which have turned data-parallelism into common culture across computing. It seems that we are all database people nowadays.

The Parallel Computing literature traditionally pooh-poohs these examples as "embarrassingly parallel." But should we really be embarrassed? Perhaps after a quarter century of fighting the "hard" problems of parallelism, the rational way forward is to start with an "easy" kernel to parallelize—something like the relational algebra—and then extend that kernel to more general-purpose computation. As PODS veterans well know, database languages have natural Turing-complete extensions (e.g., [11, 68]).

This direction for tackling parallelism and distribution raises questions that should warm the heart of a database theoretician. How does the complexity hierarchy of logic languages relate to parallel models of computation? What are appropriate complexity models for the realities of modern distributed systems, where computation is cheap and coordination is expensive? Can the lens of logic provide better focus on what is "hard" to parallelize, what is "embarrassingly easy," and what falls in between? And finally, a question close to the heart of the PODS conference: if Datalog has been The Answer all these years, is the crisis in parallel and distributed programming The Question it has been waiting for?

I explore some of these issues below, by way of both experience and conjecture.

2. BACKGROUND: DEDALUS

We work on the other side of time.
— Sun Ra

It has been seven years since my group began exploring the use of recursive queries to implement systems, based on languages including NDlog [47], Overlog [17], and SNLog [15]. But only in the last twelve months have we settled on a Datalog variant that cleanly captures what we see as the salient semantic issues for parallel and distributed computing. We call the language Dedalus, and its key contribution is the use of time as an organizing principle for distributed systems, rather than distance in physical space[3] [5]. The design of Dedalus captures the main semantic reality of distribution: two computers are effectively "distributed" if they cannot directly reason about each other's perception of time. The time dimension in Dedalus succinctly captures two important aspects of time in distributed systems: intra-node atomicity and sequencing of state transitions, and inter-node temporal relations induced by the receipt of networked data.

Dedalus clarifies many issues that were semantically ambiguous in our early work, and I will use it throughout this

[1]In tribute to Gray's twelfth challenge, our research project is called *BOOM: the Berkeley Orders Of Magnitude* project. Bloom is the language of BOOM. We hope BOOM and Bloom can be an antidote to doom and gloom.

[2]Butler Lampson filled this gap in his follow-up article in the 50^{th} anniversary issue of *J. ACM*, though he questioned the generality of declarative approaches [40].

[3]My student Peter Alvaro explains the name as follows: "Dedalus is intended as a precursor language for Bloom in the BOOM project. As such, it is derived from the character Stephen Dedalus in James Joyce's *Ulysses*, whose dense and precise chapters precede those of the novel's hero, Leopold Bloom. The character Dedalus, in turn, was partly derived from Daedalus, the greatest of the Greek engineers and father of Icarus. Unlike Overlog, which flew too close to the sun, Dedalus remains firmly grounded." [5]

paper, even for examples that predate the language. Before proceeding, I pause for a casual introduction to Dedalus targeted at database researchers familiar with Datalog.

Dedalus is a simple temporal extension to stratified Datalog in which each relation schema has a "timestamp" attribute in its rightmost position. For intuition, this attribute can be considered to contain sequence numbers from a logical clock. The use of this attribute will always be clear from context, so we can omit it from the syntax of Dedalus predicates as we will see shortly.

There are three kinds of rules in Dedalus:

- *Deductive* rules, in which all predicates share the same variable in the timestamp attribute. For such rules, we omit the timestamps completely, and the result looks like traditional Datalog. The first two rules of Figure 1 are deductive; all predicates in those rules should be considered to have a variable `T` in their missing rightmost position. Intuitively, they express deduction within each timestep.

- *Inductive* rules have the same timestamp variable in all body predicates, but the head's timestamp variable is equal to the successor of the body predicates' timestamp variable. In this case we omit the body predicates' timestamp variable, and mark the head predicate with the suffix `@next`. The third rule of Figure 1 is inductive; all the body predicates have an omitted rightmost variable `T`, the head has an omitted rightmost variable `S`, and there is an implicit body predicate `succ(T, S)`. Intuitively, this rule says that the `state` predicate at timestep `T + 1` will contain the contents of `toggle` from timestep `T`.

- *Asynchronous* rules are like inductive rules, except that the head's timestamp variable is chosen non-deterministically for each binding of the body variables in time, using Greco and Zaniolo's `choice` construct [27]. We notate asynchronous rules with the head suffix `@async`. The final rule of Figure 1 is asynchronous. It can be read akin to the inductive rule case, but with a different implicit body predicate: `choice({X, T}, {S})`, which indicates that for each pair of assignments to variables `{X, T}`, a value `S` is non-deterministically chosen. Intuitively, this syntax says that `announce` tuples are copies of `toggle` tuples, but the `announce` tuples contain (or "appear at") a non-deterministic timestep. Positive infinity is included in the domain of timestamps, corresponding to the possibility of failure in computing or communicating an asynchronous result. Most useful programs constrain the head timestamp to be larger than the body timestamp, but this is not a requirement of the language. In Section 4.2 I return to the topic of Dedalus programs that can send messages into their own past.

Dedalus includes timestamps for three reasons: to capture state visibility via timestamp unification, to capture sequential atomic update via inductive rules, and to account for the unpredictable network delays, failures and machine-clock discrepancies that occur in distributed systems via asynchronous rules. I return to these issues below, in contexts where the predecessors to Dedalus ran into difficulties.

3. EXPERIENCE

> *No practical applications of recursive query theory ... have been found to date.*
> —Michael Stonebraker, 1998
> *Readings in Database Systems, 3rd Edition*
> Stonebraker and Hellerstein, eds. [34]

Over the last seven years we have implemented and published a wide range of algorithms, protocols and complete systems specified declaratively in Datalog-like languages. These include distributed crawlers [18, 49], network routing protocols [50], overlay networks including Chord [48], distributed Bayesian inference via message passing on junction trees [8], relational query optimization [17], distributed consensus (Paxos) and two-phase commit [4], sensornet protocols [15], network caching and proxying [14, 16], file systems and job schedulers [3].

Many of these efforts were justified in terms of radical reductions in code size, typically orders of magnitude smaller than competing imperative implementations. In some cases [47, 14, 16], the results also demonstrated the advantages of automatic optimizations for declarative programs.

As a student I had little love for Datalog, and it is tempting to make amends by documenting my growing appreciation of the language and its literature. But my learning process has been slow and disorderly, and remains far from complete; certainly not a useful organizing structure for sharing the experiences from my group. Instead, this section is organized thematically. I start by describing some general behaviors and design patterns we encountered, some deficiencies of the languages we have struggled with, and implications of these for parallelism and distribution.

3.1 Recursion (Rewriting The Classics)

Our work on declarative programming began in reaction to the Web, with its emphasis on large graphs and networks. As we began working directly on this idea, we found that Datalog-style recursion had natural applications and advantages in many settings. There is no question in our minds today that 1980's-era arguments against the relevance of general recursion were short-sighted. Unfortunately, there has been too little success connecting the dots between potential and reality in this domain. Critical Web infrastructure for managing large graphs is still written in imperative languages. Closer to home, traditional RDBMS internals such as dynamic programming are also coded imperatively. Part of our agenda has been to simultaneously highlight the importance of recursion to practitioners in the database field, and to highlight the importance of declarative programming to practitioners in the systems field.

3.1.1 Finding Closure Without the Ancs

Classic discussions of Datalog start with examples of transitive closure on family trees: the dreaded `anc` and `desc` relations that afflicted a generation of graduate students[4]. My

[4]The tedium of tiresome table-names (*l'ennui de l'entité*) goes back to the founders of Datalog; the original victims can be identified by a same-generation query. However, victims often grow into abusers—a form of transitive closure—and I confess to occasional pedagogical lapses myself. This phenomenon is of course not limited to Datalog; any student of SQL can `emp`athize.

```
toggle(1) :- state(0).                    toggle(1, T) :- state(0, T).
toggle(0) :- state(1).                    toggle(0, T) :- state(1, T).
state(X)@next :- toggle(X).               state(X, S) :- toggle(X, T), succ(T, S).
announce(X)@async :- toggle(X).           announce(X, S) :- toggle(X, T), choice({X,T}, {S}).
```

Figure 1: A simple Dedalus program, written with syntactic sugar (left), and with standard Datalog notation (right).

group's work with Datalog began with the observation that more interesting examples were becoming hot topics: Web infrastructure such as webcrawlers and PageRank computation were essentially transitive closure computations, and recursive queries should simplify their implementation. To back up this claim, we began by building a Deep Web data crawler using recursive streaming SQL in the Telegraph project [18]. Subsequently we built a distributed crawler for the Gnutella peer-to-peer network as a cyclic dataflow of relational algebra in the PIER p2p query engine [49]. Both of these examples were simple monotonic programs that accumulated a list of the nodes reached from one or more starting points. We later built more sophisticated distributed programs with transitive closure at their core, including network routing protocols for Internet and wireless settings [50, 47, 15], and distributed versions of Bayesian belief propagation algorithms that pass weights along the edges [8]. As expected, Datalog was an excellent language for expressing transitive closures and graph traversals, and these tasks were almost trivial to code.

Building upon previous experience implementing RDBMS internals, my group found it relatively easy to build single-node implementations of the recursive query engines supporting these ideas. But to move to a distributed setting, two issues remained to be worked out: specification of distributed joins, and modifications to recursive query evaluation to allow asynchronous streaming of derivations across networks. These issues are discussed in Section 3.2.

3.1.2 DP and Optimization: Datalog in the Mirror

Another recursive design pattern we saw frequently was Dynamic Programming (DP). Our first work in this area was motivated by the Systems community imperative to "sip your own champagne"[5]: we wanted to implement a major part of our Overlog runtime in Overlog. To this end we built a cost-based query optimizer for Overlog named Evita Raced, itself written in Overlog as a "metacompiler" allowing for program reflection[6] [17]. Evita Raced is an Overlog program that runs in a centralized fashion without parallelism, and its DP kernel is quite similar to Greco and Zaniolo's general presentation of greedy search in extended Datalog [27]. Evita Raced makes the connection between the System R optimizer—a warhorse of the SIGMOD canon—and the compact implementation of DP in stratified Datalog. If this had been demonstrated in the 1980's during the era of extensible query optimizer architectures, it might have alleviated doubts about the utility of generalized recursion[7]. In addition to cost-based search via DP, Evita Raced also uses Overlog to implement classic Datalog optimizations and analyses, including magic sets and stratification, which are themselves based on simple transitive closures. The fact that traversals of Datalog rule/goal graphs are not described in terms of Datalog is also something of a pity, both in terms of conceptual elegance and compactness of code. But as a student of Datalog I am sypathetic to the pragmatics of exposition: it is asking a lot of one's readers to learn about recursive query optimization via metacircular logic programming[8]!

More recently, we used recursive SQL to implement the Viterbi DP algorithm for probabilistic inference on Conditional Random Fields, a technique commonly used in statistical Information Extraction [71, 70]. This connection may be more surprising to database researchers than to the machine learning community: at roughly the same time as our work on distributed systems, researchers in AI have been using logic and forward chaining to do efficient dynamic programming and search [21, 22].

Moving to a distributed setting, the main challenge that arises from Dynamic Programming is the handling of stratified aggregation (minima and maxima, etc.) across machines. I revisit this issue in Section 3.4.3.

3.2 Space, Communication and Synchrony

Much of our work has been focused on using Datalog-like languages for networking and distributed systems. This led us to a series of designs to address spatial partitioning and network communication in languages such as Overlog. Also inherent in these languages was the notion of network delay and its relationship to asynchronous evaluation.

3.2.1 Distributed State: Network Data Independence

One of our first extensions of Datalog was to model the partitioning of relations across machines in the style of parallel databases. As a matter of syntax, we required each relation to have a distinguished *location specifier* attribute. This attribute (marked with the prefix '@') had to be from the domain of network addresses in the system. Using this notation, a traditional set of network neighbor tables at each node can be represented by a global relation:

```
link(@Src, Dest, Weight)}
```

The location specifier in this declaration states that each tuple is stored at the network address of the source machine, leading to an interestingly declarative approach to networking. Location specifiers simply denote where data must be

[5]This is a more palatable (and self-congratulatory) version of the phenomenon sometimes called *dogfooding* [75].

[6]As Tyson Condie notes in his paper, the name "Evita Raced" is itself a reflection on our work: the imperfection in the name's mirroring captures the imperfect declarativity of Overlog, subsequently addressed in Dedalus.

[7]In his influential work on this topic, Guy Lohman makes an intriguing reference to logic programming, but then steps away from the idea, contrasting the deduction of "relations" from the deduction of "operators" [44].

[8]The NAIL! implementers mention using CProlog to generate rule/goal graphs for Datalog, but present imperative pseudocode for their algorithms [56].

stored; communication is induced automatically (and flexibly) to achieve this specification. As one example, consider a simple version of a multicast protocol that sends messages to designated groups of recipients:

```
received(@Node, Src, Content)@async
    :- msg(@Src, GroupID, Content),
       members(@Src, GroupID, Node).
```

The simplicity here comes from two features: representing multicast group lookup as a relational join, and the bulk specification of message shipping to all recipients via a single head variable `@Node`. Note also that this rule must be asynchronous due to the communication: we cannot say when each message will be received.

As a richer example, consider the inductive rule in path-finding:

```
path(@Src, Dest)@async
    :- link(@Src, X), path(@X, Dest).
```

Here, the unification in the body involves a variable `X` that is not a location specifier in the first predicate of the rule body; as a result, communication of some sort is required to evaluate the body. That communication can happen in at least two ways: (1) link tuples can be passed to the locations in their `X` values, and resulting join tuples shipped to the values in their `Src` attribute, or (2) the rule can be rewritten to be "left-recursive":

```
path(@Src, Dest)@async
    :- path(@Src, X), link(@X, Dest).
```

In this case path tuples can be sent to the address in their `X` attribute, joined with link tuples there, and the results returned to the address in their `Src` attribute. As it happens, evaluating these two programs in the standard left-to-right order corresponds to executing two different well-known routing protocols, one used in Internet infrastructure and another common in wireless communication [49]. Raising the abstraction of point-to-point communication to join specification leads to a wide range of optimizations for rendezvous and proxying [14, 16]. This thrust is reminiscent of the shift from navigational data models to the relational model, but in the network routing domain—an increasingly attractive idea as Internet devices and subnets evolve quickly and heterogeneously [31].

We considered this notion of Network Data Independence to be one of the key contributions we were able to bring to the Networking research community. On the other hand, as I discuss in Section 3.5.2, the global database abstraction inherent in this syntax caused us difficulty in a more general setting of distributed systems.

3.2.2 *Embracing Time, Evolving State*

Prior to the development of Dedalus, we had significant problems modeling state updates in our languages. For example, Overlog provided an operational model of persistent state with updates, including SQL-like syntax for deletion, and tuple "overwrites" via primary key specifications in head predicates. But, unlike SQL, there was no notion of transactions, and issues of update visibility were left ambiguous. The lack of clear update semantics caused us ongoing frustration, and led to multiple third-party efforts at clarifying our intended operational semantics by examining our interpreter, P2, more closely than perhaps it merited [58, 54].

```
q(V,R)@next :- q(V,R), !del_q(V,R).
qmin(V, min<R>) :- q(V,R).
p(V,R)@next :- q(V,R), qmin(V,R).
del_q(V,R) :- q(V,R), qmin(V,R).
```

Figure 2: A queue implementation in Dedalus. Predicate `q` represents the queue; items are being dequeued into a predicate `p`. Throughout, the variable `V` is a value being enqueued, and the variable `R` represents a position (or priority) in the queue.

An example of the difficulty of state update arose early in modeling the Symphony distributed hash table [53]. In Symphony, the asymptotic structure of small-world networks is simulated in practice by constraints: each new node tries to choose $\log n$ neighbors at random, subject to the constraint that no node can have more than $2 \log n$ neighbors. A simple protocol ensures this constraint: when a node wishes to join the network, it ships link establishment requests to $\log n$ randomly chosen nodes. Each recipient responds with either an agreement (establishing a bidirectional link), or a message saying it has reached its maximum degree. The recipient logic for a successful request requires a read-only check (counting the size of its neighbor table), and two updates (adding a new edge to the neighbor table, and adding a response tuple to the network stream). The check and the updates must be done in one atomic step: if two such requests are handled in an interleaved fashion at a node with $2 \log n - 1$ neighbors, they can both pass the check and lead to a violation of the maximum-degree constraint.

One solution to this "race condition" is to have the language runtime implement a queue of request messages at each recipient, dequeuing only one request at a time into the "database" considered in a given Overlog fixpoint computation. We implemented this approach in the P2 Overlog interpreter, and a similar approach is taken in the recent Reactor programming language [23]. But the use of an operational feature outside the logic is unsatisfying, and forces any program-analysis techniques to rely on operational semantics rather than model- or proof-theoretic arguments.

It is not immediately clear how to express a queue in Datalog, and our search for a suitably declarative solution to such update problems led to the design of Dedalus. The problem can be solved via the Dedalus timestamp convention, as shown in Figure 2. The first rule of Figure 2 is the Dedalus boilerplate for "persistence" via induction rather than a storage model. It asserts persistence of the head predicate `q` across consecutive timesteps, except in the presence of tuples in a designated "deletion" predicate `del_q`. The existence of a deletion tuple in timestep N breaks the induction, and the tuple no longer appears in the predicate beginning in timestep $N + 1$.

The second rule identifies the minimum item in the queue.

The third and fourth rules together atomically dequeue the minimum item in a single timestep, placing it (ephemerally) in predicate `p`. This pair of rules illustrates how multiple updates are specified to occur atomically in Dedalus. Recall that in all Dedalus rules there is an implicit but enforced unification of all body predicates on the (omitted) timestamp attribute: this enforcement of simultaneity ensures that exactly one state of the database (one timestamp) is "visible" for deduction. Inductive

rules ensure that all modifications to the state at timestep N are visible atomically in the unique successor timestep $N+1$. In Figure 2, the insertion into the relation `p` via the third rule, and the breaking of the induction in `q` via the last and first rules occur together atomically.

This queue example provides one declarative solution to operational concurrency problems in Overlog. But many other such solutions can be expressed in Dedalus. The point is that by reifying time into the language, Dedalus allows programmers to declare their desires for concurrency and isolation in the same logic that they use for other forms of deduction. Timestamp unification and the infinite successor function serve as a monotonic, stratifying construct for treating isolation at its core: as a constraint on data dependencies. We were not the first to invent this idea (Starlog and Statelog have related constructs for time and state modification [51, 42]). But we may be the most enthusiastic proponents of its utility for reasoning about parallel and distributed programming, and the role of time in computation. I return to this topic in the Conjectures of Section 4 below.

3.3 Events and Dispatch

The crux of our work has been to apply declarative database metaphors and languages to more general-purpose programs. This required us to wrestle with program features that have not traditionally been modeled in databases, including communication and task management. Both of these features fell out relatively cleanly as we developed our systems.

3.3.1 Ephemera: Timeouts, Events, and Soft State

A central issue in distributed systems is the inability to establish the state of a remote machine. To address this limitation, distributed systems maintain evidence of the liveness of disparate nodes via periodic "heartbeat" messages and "timeouts" on those messages. This design pattern has been part of every distributed algorithm and system we have built. To support it, we needed our languages to capture the notion of physical clocks at each node.

Early on, we incorporated the notion of physical time as a relation in our languages. In Overlog we provided a built-in predicate `periodic` that could be declared to contain a set (possibly infinite) of tuples with regularly-spaced wall-clock times; the P2 runtime for Overlog would cause these tuples to "appear" in the dataflow of the query engine at the wall-clock times they contained. We would use this predicate to drive subsequent tuple derivations, for example the generation of scheduled heartbeat messages. Tuples in Overlog's `periodic` table were intended to be "ephemeral events", formed to kick off a computation once, and then be "forgotten."

Ephemeral state is often desired in network protocols, where messages are "handled" and then "consumed." A related pattern in networking is *soft state* maintenance, a loosely-coupled protocol for maintaining caches or views across machines. Consider a situation where a receiver node is caching objects (e.g., routing table entries) known at some sender node. The sender and receiver agree upon a time-to-live (TTL) for this "soft" state. The sender must try to send "refresh" messages—essentially, per-object heartbeat messages—to the receiver before the TTL expires. Upon receipt, the receiver resets the TTL of the relevant object to the agreed-upon maximum; in the absence of a refresh within the TTL, the receiver deletes the object.

While these are common patterns in networking, their inclusion in Overlog complicated our language semantics. We included persistence properties as an aspect of Overlog's table declaration: tables could be marked as persistent, soft-state (with a fixed TTL) or as event tables (data streams). This feature introduced various subtleties for rules that mixed persistent predicates with soft state or event predicates [45]. Consider an Overlog rule for logging events: it has a persistent table in the head that represents the log, and an ephemeral stream of events in the body. But for such an Overlog rule, what does it mean when an ephemeral body tuple "disappears"? We would like the logged tuple in the head to remain, but it is no longer supported by an existing body fact.

Dedalus clears up the confusion by treating *all* tuples as ephemeral "events." Persistence of a table is ensured by the deduction of new (ephemeral) tuples at each timestep from the same tuples at the preceding timestep, as in the first rule of Figure 2 above[9]. Ambiguous "race conditions" are removed by enforcing the convention of unification on timestamp attributes. Soft state can be achieved by modifying the persistence rules to capture a wall-clock time attribute of tuples in soft-state tables (via join with a built-in wallclock-time relation), and by including a TTL-checking clause in the persistence rule as follows:

```
q(A, TTL, Birth)@next :-
   q(A, TTL, Birth),!del_q(A),
   now() - Birth < TTL.
```

In this example, `now()` returns the current wall-clock time at the local node, and can be implement as a foreign function in the spirit of LDL [13].

Having reified time into an attribute in Dedalus, any ambiguities about persistence that were inherent in Overlog are required to be explicit in a programmer's Dedalus specification. There is no need to resort to operational semantics to explain why a tuple "persists," "disappears," or is "overwritten" at a given time: each Dedalus tuple is grounded in its provenance. All issues of state mutation and persistence are captured within that logical framework.

3.3.2 Dispatch as Join: A Third Way

At the heart of any long-running service or system is a dispatching model for the management of multiple tasks. There are two foundational design patterns for task dispatch: concurrent processes and event loops. In a classic paper, Lauer and Needham demonstrate a duality between these patterns [41], but in applied settings in the last decade there has been significant back-and-forth on the performance superiority of one model or the other (e.g., [72, 69]).

We found ourselves handling this issue with a third design pattern based on dataflow. Our crawlers specified dispatch via the streaming join of an event table and a persistent system-state table. To illustrate, consider the simple example of Figure 3, which handles service `request` tuples with parameter `P`. At the timestep when a particular request arrives, it is recorded in the `pending` requests table, where it persists until it receives a matching `response`. The invocation of `service_in` is specified to happen at the same atomic timestep as the request arrival; it is evaluated by an asynchronous external function call that will eventually place its results in the

[9] An intelligent evaluation strategy for this logic should in most cases use traditional storage technology rather than re-deriving tuples each timestep.

```
pending(Id, Sender, P) :-
    request(Id, Sender, P).
pending(Id, Sender, P)@next :-
    pending(Id, Sender, P),
    !response(Id, Sender, _).
service_out(P, Out)@async :-
    request(Id, Sender, P),
    service_in(P, Out).
response(Sender, Id, O) :-
    pending(Id, Sender, P),
    service_out(P, O).
```

Figure 3: An asynchronous service

relation `service_out`. Because this computation is asynchronous, the system need not "wait" for results before beginning the next timestep. This approach follows the common model of lightweight event handlers. As `service_out` results arrive (likely in a different order than their input), they need to be looked up in the "rendezvous buffer" of `pending` requests to be routed back to the caller.

Evaluating this logic requires nothing more than the execution of a number of pipelined join algorithms such as that of Wilschut and Apers [76]. The application of pipelining joins to asynchronous dispatch was first explored in database literature for querying remote services on the Web [25, 63]. But the implication is much broader: any server dispatch loop can be coded as a few simple joins in a high-level language. This data-centric approach parallelizes beautifully (using a hash of `Id` as a location specifier), it does not incur the context-switching overhead associated with the process model, nor does it require the programmer to write explicit logic for state loops, event handling, and request-response rendezvous.

Moreover, by writing even low-level tasks such as request dispatch in logic, more of the system can enjoy the benefits of higher level reasoning, including simplicity of specification, and automatic query optimization across multiple software layers. For example, application logic that filters messages can be automatically merged into the scheduler via "selection pushdown" or magic sets rewrites, achieving something akin to kernel packet filters without any programmer intervention. Scheduling can be easily spread across multiple nodes, with task and data partitioning aligned for locality in a straightforward manner. It has yet to be demonstrated that the inner loop of a dispatcher built on a query engine can compete for performance with the best implementations of threads or events. I believe this is achievable. I also believe that optimization and parallelization opportunities that fall out from the data-centric approach can make it substantially out-perform the best thread and event packages.

3.4 Parallel and Distributed Implications

Having discussed our experience with these design patterns in some detail, I would like to step back and evaluate the implications for Datalog-like languages in parallel and distributed settings.

3.4.1 Monotonic? Embarrassing!

The Pipelined Semi-Naive (PSN) evaluation strategy [47] lies at the heart of our experience running Datalog in parallel and distributed settings. The intuition for PSN comes from our crawler experience. The basic idea in the crawlers was to act immediately on the discovery a new edge in two ways: add its endpoints to the table of nodes seen so far, and if either endpoint is new, send a request to probe that node for its neighbors in turn, producing more new edges.

It should be clear that this approach produces a correct traversal of the network graph regardless of the order of node visits. But it is a departure from classical semi-naive evaluation, which proceeds in strict rounds corresponding to a breadth-first traversal of the graph. The need to wait for each level of the traversal to complete before moving on to the next requires undesirable (unacceptable!) coordination overhead in a distributed or parallel setting. Moreover, it is unnecessary: in monotonic programs, deductions can only "accumulate," and need never be postponed. PSN makes this idea work for general monotonic Datalog programs, avoiding redundant work via a sequencing scheme borrowed from the Urhan-Franklin Xjoin algorithm [65]. The result is that monotonic (sub)programs can proceed without any synchronization between individual deductions, and without any redundant derivations. Simply put, PSN makes monotonic logic embarrassingly parallel. This statement is significant: a large class of recursive programs—all of basic Datalog—can be parallelized without any need for coordination! This simple point is at the core of the Conjectures in Section 4 below.

As a side note, this insight appears to have eluded the MapReduce community as well, where join is necessarily a blocking operator. The issue that arises in MapReduce results from an improper conflation of a physical operation (repartitioning data) with a non-monotonic functional operation (Reduce). In Google's MapReduce framework, the only way to achieve physical network repartitioning—a key component to parallel joins—is to use a Reducer. The framework assumes Reducers need to see all their inputs at once, so it introduces a parallel barrier: no node in the system may begin Reducing until all the Map tasks are complete. This defeats the pipelining approach of Wilschut and Apers, which would otherwise perform the monotonic logic of join using physical network partitioning as a primitive. A clean implementation should be able to choose between the efficiency of pipelining and the simple fault-tolerance that comes from materialization, without tying the decision unnecessarily to the programming model.

3.4.2 Monotonic? Eventually Consistent!

A related feature we exploited in our crawlers was to accomodate "insertions" to the database via simple ongoing execution of PSN evaluation. The goal, formalized by Loo [47], was to have an *eventually consistent* semantics for the links and paths in the graph: in a quiescent database without communication failure, the set of derived data across all machines should eventually reach a consistent state. It is easy to achieve this consistency for monotonic insertion of edges into a crawler. When a new edge is added, paths in the graph radiating out from the new edge can be crawled and added to the transitive closure. When edges cease to arrive, this process eventually leads to a consistent transitive closure. This approach can be seen as a materialized view scheme for transitive closure, but in effect it is no different than the *de novo* PSN query evaluation scheme sketched above: updates simply play the role of edges that "appear very late" in the evaluation strategy.

More generally, "monotonic updates" (i.e. "appends") to a

monotonic program guarantee eventual consistency. And this result can be achieved without any redundant work, by simply leaving the standard pipelined query-processing algorithm running indefinitely.

Note that in Dedalus, non-monotonic updates (deletion, overwriting) are expressed by non-monotonic programs: the negated `del` clause in a persistence rule such as the one in Figure 2. A monotonic Dedalus program can persist appends via a simpler rule:

```
p(X)@next :- p(X).
```

But modeling deletion or overwrite requires negation. Hence using Dedalus we can simply speak of whether a program is monotonic or not; this description includes the monotonicity of its state manipulation. This point informs much of the discussion in Section 4.

3.4.3 Counting Waits; Waiting Counts

If coordination is not required for monotonic programs, when is it required? The answer should be clear: at non-monotonic stratification boundaries. To establish the veracity of a negated predicate in a distributed setting, an evaluation strategy has to start "counting to 0" to determine emptiness, and wait until the distributed counting process has definitely terminated. Aggregation is the generalization of this idea.

It is tricky to compute aggregates in a distributed system that can include network delays, reordering, and failures. This problem has been the topic of significant attention in the last decade [52, 66, 36, 57] etc.) For recursive strata that span machines, the task is even trickier: no node can establish in isolation that it has fully "consumed" its input, since recursive deductions may be in flight from elsewhere.

In order to compute the outcome of an aggregate, nodes must wait and coordinate. And the logic of the next stratum of the program must wait until the coordination is complete: in general, no node may start stratum $N+1$ until all nodes are known to have completed stratum N. In parallel programming parlance, a stratification boundary is a "global barrier." More colloquially, we can say that counting requires waiting.

This idea can be seen from the other direction as well. Coordination protocols are themselves aggregations, since they entail voting: Two-Phase Commit requires unanimous votes, Paxos consensus requires majority votes, and Byzantine protocols require a 2/3 majority. Waiting requires counting.

Combining this discussion with the previous two observations, it should be clear that there is a deep connection between non-monotonic reasoning and parallel coordination. Monotonic reasoning can be done without any coordination among nodes; non-monotonic reasoning in general requires global barriers. This line of thought suggests that non-monotonicity—a property of logic programs that can sometimes be easily identified statically—is key to understanding the limits of parallelization. I return to this point in the Conjectures section.

3.4.4 Unstratifiable? Spend Some Time.

The Dedalus state-update examples presented earlier show how the sequentiality of time can be used to capture atomic updates and persistence. Time can also be used to make sense of otherwise ambiguous, unstratifiable programs. Consider the following program, a variation on Figure 1 that toggles the emptiness of a predicate:

```
state(X)@next :- state(X), !del_state(X).
state(1) :- !state(X).
del_state(X) :- state(X)
```

The first rule is boilerplate Dedalus persistence. The last two rules toggle the emptiness of state. The second rule is clearly not stratifiable. But if we make the second rule inductive, things change:

```
state(1)@next :- !state(X).
```

In this revised program, the state table only depends negatively on itself across timesteps, never within a single timestep. The resulting program has a unique minimal model, which has `1` in the state relation every other timestep.[10]

If we expand the syntax of this Dedalus program with all the omitted attributes and clauses, we can see that it provides what Ross defined as *Universal Constraint Stratification* [64] by virtue of the semantics of the successor function used in time. Universal Constraint Stratification is a technique to establish the acyclicity of individual derivations by manipulating constraints on the semantics of functions in a program. In this program, the successor function ensures that all derivations of state produce monotonically increasing timesteps, and hence no derivation can cycle through negation.

Many programs we have written—including the queue example above—are meaningful only because time flies like an arrow: monotonically forward[11]. Again, this temporal construct hints at a deeper point that I will expand upon in Section 4.3: in some cases the meaning of a program can only be established by "spending time."

3.5 Gripes and Problems

Datalog-based languages have enabled my students to be very productive coders. That said, it is not the case that they have always been happy with the languages at hand. Here I mention some of the common complaints, with an eye toward improving them in Bloom.

3.5.1 Syntax and Encapsulation

The first frustration programmers have with Datalog is the difficulty of unifying predicates "by eyeball," especially for predicates of high arity. Off-by-one errors in variable positions are easy to make, hard to recognize, and harder to debug. Code becomes burdensome to read and understand because of the effort involved in visually matching the index of multiple variables in multiple lists.

Datalog often requires redundant code. Disjunction, in our Datalog variants, involves writing multiple rules with the same head predicate. Conditional logic is worse. Consider the following example comparing a view of the number of "yes" votes to a view of the total number of votes:

```
outcome('succeed')
    :- yes(X), total(Y), X > Y/2.
outcome('fail')
    :- yes(X), total(Y), X < Y/2.
```

[10] Technically, the minimal model here is infinitely periodic, but with a minimal number of distinguished states (two). Capturing this point requires a slightly modified notion of safety and minimality [42].

[11] Groucho Marx's corollary comes to mind: "Time flies like an arrow. Fruit flies like a banana."

```
outcome('tie')
    :- yes(X), total(Y), X = Y/2.
```

Not only is this code irritatingly chatty, but the different "branches" of this conditional expression are independent, and need not appear near each other in the program text. Only programmer discipline can ensure that such branches are easy to read and maintain over the lifetime of a piece of software.

Finally, Datalog offers none of the common constructs for modularity: variable scoping, interface specification, encapsulation, etc. The absence of these constructs often leads to disorganized, redundant code that is hard to read and evolve.

Many of these gripes are addressable with syntactic sugar, and some Datalog variants offer help [6]. One observation then is that such sugar is very important in practice, and the academic syntax of Datalog has not improved its odds of adoption.

3.5.2 The Myth of the Global Database

The routing examples discussed above illustrate how communication can be induced via a partitioned global database. The metaphor of a global database becomes problematic when we consider programs in which semantics under failure are important. In practice, individual participating machines may become disconnected from and reconnected to the network over time, taking their partitions with them. Moreover time (and hence "state") may evolve at different rates on different nodes. Exposing a unified global database abstraction to the programmer is therefore a lie. In the context of routing it is a little white lie, because computation of the true "best" paths in the network at any time is not practically achievable in any language, and not considered important to the task. But the lie can cause trouble in cases where assumptions about global state affect correctness. False abstractions in distributed protocols have historically been quite problematic [77]. In our recent Overlog code we have rarely written rules with distributed joins in the body, in part because it seems like bad programming style, and in part because we have been focused on distributed systems protocols where message failure handling needs to be explicit in the logic. In Dedalus such rules are forbidden.

Note that the myth of the global database can be "made true" via additional code. We have implemented distributed consensus protocols such as Two-Phase Commit and Paxos that can ensure consistent, network-global updates. These protocols slow down a distributed system substantially, but in cases where it is important, distributed joins can be made "real" by incorporating these protocols [4]. On top of these protocols, an abstraction of a consistent global database can be made true (though unavailable in the face of network partitions, as pointed out in Brewer's CAP theorem.)

Given that distributed state semantics are fundamental to parallel programming, it seems important for the language syntax to require programmers to address it, and the language parser to provide built-in reasoning about the use of different storage types. For example, the Dedalus boilerplate for persistence can be "sugared" via a persistence modifier to a schema declaration, as in Overlog. Similarly, the rules for a persistent distributed table protected via two-phase commit updates could be sugared via a "globally consistent" schema modifier. While these may seem like syntactic sugar, from a programmer's perspective these are critical metaphors: the choice of the proper storage semantics can determine the meaning and efficiency of a distributed system. Meanwhile, the fact that all these options compile down to Dedalus suggests that program analysis can be done to capture the stated meaning of the program and reflect it to the user. In the other direction, program analysis can in some cases relax the user's choice of consistency models without changing program semantics.

4. CONJECTURES

> *In placid hours well-pleased we dream*
> *Of many a brave unbodied scheme.*
> — Herman Melville

The experiences described above are offered as lessons of construction. But a database theory audience may prefer the construction of more questions. Are there larger theoretical issues that arise here, and can they inform practice in a fundamental way?

I like to think the answer is "yes," though I am sensitive to both the hubris and irresponsibility of making up problems for other people. As a start, I offer some conjectures that have arisen from discussion in my group. Perhaps the wider database theory audience will find aspects of them amenable to formalization, and worthy of deeper investigation.

4.1 Parallelism, Distribution and Monotonicity

Earlier I asserted that basic Datalog programs—monotonic programs without negation or aggregation—can be implemented in an embarrassingly parallel or eventually consistent manner without need for any coordination. As a matter of conjecture, it seems that the other direction should hold as well:

CONJECTURE 1. Consistency And Logical Monotonicity (CALM). *A program has an eventually consistent, coordination-free execution strategy if and only if it is expressible in (monotonic) Datalog.*

The "coordination-free" property is key to the CALM conjecture. Clearly one can achieve eventual consistency via a coordination mechanism such as two-phase commit or Paxos. But this "instantaneous consistency" approach violates the spirit of eventual-consistency methods, which typically proceed without coordination and still produce consistent states in periods of quiescence.

I have yet to argue one direction of this conjecture: that a non-monotonic program can have no eventually consistent implementation without coordination. Consider the case of a two-stratum non-monotonic program and some eventually consistent implementation. Any node in the system can begin evaluating the second stratum only when it can prove it has received "everything" in the first stratum's predicates. For global consistency, "everything" in this context means any data that is interdependent with what any other node has received. If any of that data resides on remote nodes, distributed coordination is required.

A proper treatment of this conjecture requires crisp definitions of eventual consistency, coordination, and relevant data dependencies. In particular, trivially partitionable programs with no cross-node interdependencies need to be treated as a special (easy) case. But I suspect the conjecture holds for most

practical cases of interest in distributed and parallel computing.

It is worth recalling here Vardi's well-known result that (monotonic) Datalog can be evaluated in time polynomial in the size of the database [67]. If the conjecture above is true, then the expressive power of "eventually-consistent" implementations is similarly bounded, where the "database" includes all data and messages introduced up to the time of quiescence.

This conjecture, if true, would have both analytic and constructive uses. On the analytic front, existing systems that offer eventual consistency can be modeled in Datalog and checked for monotonicity. In many cases the core logic will be trivially monotonic, but with special-purpose escapes into non-monotonicity that should either be "protected" by coordination, or managed via compensatory exception handling (Helland and Campbell's "apologies" [30].) As a classic example, general ledger entries (debits and credits) accumulate monotonically, but account balance computation is non-monotonic; Amazon uses a (mostly) eventually-consistent implementation for this pattern in their shopping carts [19]. An interesting direction here is to incorporate exception-handling logic into the program analysis: ideally, a program with proper exception handlers can be shown to be monotonic even though it would not b monotonic in the absence of the handlers. Even more interesting is the prospect of automatically generating (perhaps conservative) exception handlers to enforce a provable notion of monotonicity.

On the constructive front, implementations in Datalog-style languages should be amenable to (semi-)automatic rewriting techniques that expose further monotonicity, expanding the base of highly-available, eventually-consistent functionality. As an example, a predicate testing for non-negative account balances can be evaluated without coordination for accounts that see only credit entries, since the predicate will eventually transition to truth at each node as information propagates, and will never fail thereafter. This example is simplistic, but the idea can be used in a more fine-grained manner to enable sets or time periods of safe operation (e.g., a certain number of "small" debits) to run in an eventually consistent fashion, only enforcing barriers as special-case logic near monotonicity thresholds on predicates (e.g., when an account balance is too low to accomodate worst-case scenarios of in-flight debits). It would be interesting to enable program annotations, in the spirit of Universal Constraint Stratification [64], that would allow program rewrites to relax initially non-monotonic kernels in this fashion.

Finally, confirmation of this conjecture would shed some much-needed light on heated discussions of the day regarding the utility or necessity of non-transactional but eventually consistent systems, including the so-called "NoSQL" movement. It is sorely tempting to underscore this conjecture with the slogan "NoSQL is Datalog." But my student Neil Conway views this idea as pure mischief-making, so I have refrained from including the slogan here.

4.2 Asynchrony, Traces, and Trace Equivalence

Expanding on the previous point, consider asynchronous rules, which introduce non-determinism into Dedalus timestamps and hence Dedalus semantics. It is natural to ask under what conditions this explicit non-determinism affects program outcomes, and how.

We can say that the timestamps in asynchronous head predicates capture possible *traces* of a program: each trace is an assignment of timestamps that describes a non-deterministic "run" of an evaluation strategy. We can define *trace equivalence* with respect to a given program: two traces can be considered equivalent if they lead to the same "final" outcome of the database modulo timestamp attributes. If all traces of a program can be shown to be equivalent in this sense, we have demonstrated the Church-Rosser confluence property. In cases where this property does not hold, other notions of equivalence classes may be of interest. Serializability theory provides a model: we might try to prove that every trace of a program is in an equivalence class with some "good" trace.

The theory of distributed systems has developed various techniques to discuss the possible traces of a program. The seminal work is Lamport's paper on "Time, Clocks and the Ordering of Events" [39], which lays out the notion of causal ordering in time that requires logical clocks to respect a *happens-before* relation. Based on these observations, he shows that multiple independent clocks (at different distributed nodes) can be made to respect the happens-before relation of each individual node. Coordination protocols have been developed to enforce this kind of synchrony, and related issues involving data consistency. We have coded some of these protocols in Dedalus and its predecessor languages [4], and they can be incorporated into programs to constrain the class of traces that can be produced.

Classical PODC work on casuality tends to assume black-box state machines at the communication endpoints. With Dedalus programs at the endpoints, we can extract logical data dependency and provenance properties of the programs, including tests for various forms of stratifiability. Can the combination of causality analysis and logic-programming tests enrich our understanding of distributed systems, and perhaps admit new program-specific cleverness in coordination?

As an extreme example, suppose we ignore causality entirely, and allow asynchronous Dedalus rules to send messages into the past. Are temporal paradoxes—the absence of a unique minimal model—an inevitable result? On this front, I have a simple conjecture:

CONJECTURE 2. Causality Required Only for Non-monotonicity (CRON). *Program semantics require causal message ordering if and only if the messages participate in non-monotonic derivations.*

Said differently, temporal paradoxes arise from messages sent into the past if and only if the messages have non-monotonic implications.

This conjecture follows the intuition of the CALM Conjecture. Purely monotonic logic does not depend on message ordering, but if the facts being "sent into the past" are part of a non-monotonic cycle of deduction, the program lacks a unique minimal model: it will either admit multiple possible worlds, or none.

The idea of sending messages into the past may seem esoteric, but it arises in practical techniques like recovery. If a node fails and is restarted at time T, it may reuse results from logs that recorded the (partial) output of a run of the same logic from some earlier time $S < T$. In effect the derived messages can "appear" at time S, and be used in the redo execution beginning at T without causing problems. Speculative

execution strategies have a similar flavor, a point I return to in Section 4.4.

Exploiting the monotonic case may be constructive. It should be possible to relax the scheduling and spatial partitioning of programs—allow for more asynchrony via a broader class of traces—by examining the program logic, and enforcing causal orderings only to control non-monotonic reasoning. This marriage of PODS-style program analysis and PODC-style causality analysis has many attractions.

4.3 Coordination Complexity, Time and Fate

In recent years, the exponentiation of Moore's Law has brought the cost of computational units so low that to infrastructure services they seem almost free. For example, O'Malley and Murthy at Yahoo! reported sorting a petabyte of data with Hadoop using a cluster of 3,800 machines each with 8 processor cores, 4 disks, and 8GB of RAM each [59]. That means each core was responsible for sorting only about 32 MB (just 1/64th of their available share of RAM!), while 3799/3800 of the petabyte was passed through the cluster interconnection network during repartitioning. In rough terms, they maximized parallelism while ignoring resource utilization. But if computation and communication are nearly free in practice, what kind of complexity model captures the practical constraints of modern datacenters?

This anecdote involves an embarrassingly parallel, monotonic binning algorithm, with a focus on bulk data throughput rather than latency of individual interactions. By contrast, non-monotonic stratified programs require latency-sensitive distributed coordination to proceed from one parallel monotonic stratum to the next. Non-monotonic stratum boundaries are global barriers: in general, no node can proceed until all tasks in the lower stratum are guaranteed to be finished. This restriction means that the slowest-performing task in the cluster—the "weakest link"— slows all nodes down to its speed. Dynamic load balancing and reassignment can mitigate the worst-case performance of the weakest link, but even with those techniques in place, coordination is the key remaining bottleneck in a world of free computation [9].

In this worldview, the running time of a logic program might be best measured by the number of strata it must proceed through sequentially; call this the Coordination Complexity of a program[12]. This notion differs from other recent models of parallel complexity for MapReduce proposed by Afrati and Ullman [2] and Karloff, et al. [38], which still concern themselves in large part with measuring communication and computation. It resembles a simplified ("embarrassingly" simplified?) form of Valiant's Bulk Synchronous-Parallel (BSP) model, with the weights for communication and computation time set to zero. Results for BSP tend to involve complicated analyses of communication and computation metrics that are treated as irrelevant here, with good reason. First, the core operations for bottom-up Dedalus evaluation (join, aggregation) typically require all-to-all communication that only varies between $\frac{1}{2}$ and 1 for any non-trivial degree of parallelism[13]. Second, at scale, the practical running time of the slowest node in the cluster is often governed less by computational complexity than by non-deterministic effects of the environment (heterogeneous machines and workloads, machine failures, software misconfiguration and bugs, etc.)

Conveniently, a narrow focus on Coordination Complexity fits nicely with logic programming techniques, particularly if the previous conjectures hold: i.e., if coordination is required precisely to manage non-monotonic boundaries. In that case, the Coordination Complexity of a stratified program is the maximum stratum number in the program, which can be analyzed syntactically. In the more general case of locally stratified [62] or universally constraint-stratified programs [64], the program's rule-goal graph may have cycles through negation or aggregation, which in practice might be traversed many times. The coordination complexity in these cases depends not only on the rule syntax but also on the database instance (in the case of local stratification) and on the semantics of monotonic predicates and aggregations in the program (in universal constraint stratification).

As noted above, many natural Dedalus programs are not stratified, but are instead universally constraint-stratified by the monotonicity of timestamps. In those cases, the number of times around the non-monotonic loops corresponds to the number of Dedalus timesteps needed to complete the computation. In essence, Dedalus timesteps become units for measuring the complexity of a parallel algorithm.

This idea is intuitively appealing in the following sense. Recall that a Dedalus timestep results in an atomic batch of inductions, corresponding to traditional "state changes." To lower the number of Dedalus timesteps for a program, one needs to find a way to batch together more state modifications within a single timestep—i.e., shift some rules from inductive to deductive (by removing the `@next` suffix). If a program is expressed to use a minimal number of timesteps, it has reached its inherent limit on "batching up" state changes—or conversely, the program accurately captures the inherent need for sequentiality of its state modifications.

This argument leads to our next conjecture:

CONJECTURE 3. Dedalus Time ⇔ Coordination Complexity. *The minimum number of Dedalus timesteps required to evaluate a program on a given input data set is equivalent to the program's Coordination Complexity.*

The equivalence posited here is within a constant factor of the minimum number of sequential coordination steps required, which accounts for multiple deductive strata within a single timestep. The stratification depth per timestep is bounded by the program's length, and hence is a constant with respect to data complexity (the appropriate measure for analyzing a specific program [67]).[14]

Clearly one can do a poor job writing an algorithm in Dedalus, e.g., by overuse of the `@next` modifier. So when are timesteps truly required? We can distinguish two cases. The first is when the removal of an `@next` suffix changes the meaning

[12] In some algorithms it may be worth refining this further to capture the fraction of nodes involved in each coordination step; this two-dimensional "depth" and "breadth" might be called a coordination *surface* or lattice.

[13] The exception here is cases where communication can be "colored away" entirely, due to repeated partitioning by functionally dependent keys [29].

[14] David Maier notes that it should be possible convert a fixed number of timestamps into data and account for these timestamps within a single Dedalus timestep, in the manner of loop unrolling. This conversion of time into space may require refining the complexity measure proposed here.

(i.e. the minimal model) of the program. The second is when the removal causes a form of non-monotonicity that leaves the program with no unique minimal model: either a proliferation of minimal models, or no models because a contradiction requires some variable to take on multiple values "at once." The examples we have seen for the first of these cases seem trivial, in the spirit of Figure 1: a finite number of possible states are infinitely repeated. Such programs can be rewritten without an infinite successor relation: the finite set of possible states can be distinguished via data-oriented predicates on finite domains, rather than contemporaneity in unbounded (but cyclic) time[15]. This leads us to the following more aggressive conjecture:

CONJECTURE 4. Fateful Time. *Any Dedalus program P can be rewritten into an equivalent* temporally-minimized *program P' such that each inductive or asynchronous rule of P' is* necessary*: converting that rule to a deductive rule would result in a program with no unique minimal model.*

I call this the "Fateful Time" conjecture because it argues that *the inherent purpose of time is to seal fate*. If a program's semantics inherently rely upon time as an unbounded source of monotonicity, then the requirement for simultaneity in unification resolves multiple concurrent possible worlds into a series of irrevocable individual worlds, one at each timestep. If the conjecture holds, then any other use of temporal induction is literally a waste of time.[16].

This conjecture is an appealing counterpart to CRON. Time *does* matter, exactly in those cases where ignoring it would result in logically ambiguous fate[17].

[15]This observation, by my student William Marczak, is eerily reminiscent of the philosophy underlying Jorge Luis Borges' stories of circular ruins, weary immortals, infinite libraries and labyrinths. In his "New Refutation of Time," Borges presents a temporal extension of the idealism of Berkeley and Hume (which denies the existence of matter) based on the following argument: "The denial of time involves two negations: the negation of the succession of the terms of a series, negation of the synchronism of the terms in two different series." The similarity to a rewriting of Dedalus' successors and timestamp unification is striking. More allegorically, Borges puts it this way: "I suspect, however, that the number of circumstantial variants is not infinite: we can postulate, in the mind of an individual (or of two individuals who do not know of each other but in whom the same process works), two identical moments. Once this identity is postulated, one may ask: Are not these identical moments the same? Is not one single repeated term sufficient to break down and confuse the series of time? Do not the fervent readers who surrender themselves to Shakespeare become, literally, Shakespeare?" [10]

[16]Temporally-minimized Dedalus programs might be called *daidala*, Homer's term for finely crafted basic objects: "The 'daidala' in Homer seem to possess mysterious powers. They are luminous—they reveal the reality that they represent" [61].

[17]In his Refutation, Borges concludes with a very similar inescapable association of time and fate: "*And yet, and yet...* Denying temporal succession, denying the self, denying the astronomical universe are apparent desperations and secret consolations. Our destiny ... is frightful because it is irreversible and iron-clad. Time is the substance I am made of. Time is a river which sweeps me along, but I am the river; it is a tiger which destroys me, but I am the tiger; it is a fire which consumes me, but I am the fire. The world, unfortunately, is real; I, unfortunately, am Borges" [10]. To close the cycle here, note that Daedalus was the architect of the Labyrinth of Crete; labyrinths are a signature metaphor in Borges' writing.

4.4 Approximation and Speculation

Given the cost of coordination at stratum boundaries, it is tempting to try and go further: let the evaluation of a Dedalus program press ahead without waiting for the completion of a stratum or timestep. In some cases this trick can be done safely: for example, temporal aggregates such as `count` and `min` can provide "early returns" in the spirit of online aggregation [33], and range predicates on the results of those aggregates can sometimes be evaluated correctly in parallel with the computation of the final aggregate result [78]. These cases exploit monotonicity of predicates on monotonic aggregates, and are in the spirit of the CALM conjecture above.

But what about approximable but non-monotonic aggregates, such as averages, which can produce early estimates that oscillate non-monotonically, but provide probabilistic confidence intervals? If predicates on the result of those aggregates are "acted upon" probabilistically, in parallel with the completion of the lower stratum, what happens to the program outcome?

Two execution strategies come to mind, based on "optimistically" allowing higher strata to proceed on the basis of tentative results in lower strata. The first takes the tentative results and acts upon them directly to compute a fixpoint, which may or may not be the same as the minimal model of the program. The challenge then is to characterize the distribution of possible worlds that can arise from such evaluations, provide a meaningful probabilistic interpretation on the outcomes of the computation, and perhaps provide execution strategies to ensure that an individual outcome has high likelihood. It would be interesting to understand how this relates to more traditional approximation algorithms and complexity, especially with respect to parallel computation.

A second strategy is to ensure the correct minimal model by "rolling back" any deductions based on false assumptions using view maintenance techniques [12]. Here the challenge is not to characterize the answer, but to produce it efficiently by making good guesses. This requires characterizing the expected utility of a given optimistic decision, both in terms of its likely benefit if performance is correct, and its likely recomputation cost if incorrect. This is in the spirit of speculation techniques that are common currency in the Architecture and Systems communities, but with the benefit of program analyses provided by logic. It also seems feasible to synthesize logic here, including both speculative moves, and compensating actions for incorrect speculations.

Practically, these approaches are important for getting around fundamental latencies in communication. From a theoretical perspective, speculation on non-monotonic boundaries seems to be the natural path to bring randomized algorithms into the evaluation of logic: the previous conjectures suggest that there is no interesting non-determinism in monotonic programs, so the power of randomized execution seems to reside at non-monotonic boundaries. It would be interesting to understand how to bridge this idea to the complexity structures known for randomized algorithms.

This line of thinking is not as well developed as the earlier discussion, so I close this discussion without stating a particular conjecture.

5. CARPE DIEM

Gather ye rosebuds while ye may
Old Time is still a-flying,
And this same flower that smiles to-day
To-morrow will be dying.
— Robert Herrick

Like most invited papers, I would have rejected this one had I been asked to review it[18]. Its claims are imprecise and unsubstantiated, relying on anecdotes and intuition. It contains an unseemly number of references to the author's prior work. It is too long, and was submitted after the conference deadline. And as a paper that treads outside the author's area of expertise, it also undoubtedly overlooks important earlier work by others. For this last flaw in particular I extend sincere apologies, and an earnest request to be schooled regarding my omissions and errors.

However, my chief ambition in writing this paper was not to present a scholarly survey. It was instead to underscore—in as urgent and ambitious terms as possible—the current opportunity for database theory to have broad impact. Under most circumstances it is very hard to change the way people program computers [55]. But as noted by Hennessy and others, programming is entering an unusually dark period, with dire implications for computer science in general. "Data folk" seem to have one of the best sources of light: we have years of success parallelizing SQL, we have the common culture of MapReduce as a bridge to colleagues, and we have the well-tended garden of declarative logic languages to transplant into practice.

Circumstance has presented a rare opportunity—call it an imperative—for the database community to take its place in the sun, and help create a new environment for parallel and distributed computation to flourish. I hope that the discussion in this paper will encourage more work in that direction.

6. ACKNOWLEDGMENTS

I am indebted to mentors who read and commented on early drafts of this paper: David Maier, Jeff Naughton, Christos Papadimitriou, Raghu Ramakrishnan, Jeff Ullman, and Moshe Vardi. I am grateful for their time, logic, and bracing candor. I am also grateful to the organizers of PODS 2010 for their invitation to prepare this material.

The conjectures in the paper grew out of relatively recent discussions with students on the BOOM project: Peter Alvaro, Tyson Condie, Neil Conway, William Marczak and Rusty Sears. I am unreasonably lucky and genuinely grateful to be able to work with a group of students this strong.

Much of the fun of systems work comes from the group effort involved. In certain cases in the text I was able to give specific references to ideas from individuals, but many of the experiences I described arose in joint work with a wonderful group of students and colleagues, especially on the P2 [46], DSN [15] and BOOM [3] projects. The following list is the best accounting I can give for these experiences. In alphabetical order, credit is due to: Peter Alvaro, Ashima Atul, Ras Bodik, Kuang Chen, Owen Cooper, David Chu, Tyson Condie, Neil Conway, Khaled Elmeleegy, Stanislav Funiak, Minos Garofalakis, David Gay, Carlos Guestrin, Thibaud Hottelier, Ryan Huebsch, Philip Levis, Boon Thau Loo, David Maier, Petros Maniatis, Lucian Popa, Raghu Ramakrishnan, Timothy Roscoe, Rusty Sears, Scott Shenker, Ion Stoica, and Arsalan Tavakoli.

[18]My benchmark of quality for invited papers is Papadimitriou's "Database Metatheory" [60]. Although I looked to it for inspiration, I had no pretensions to its breadth or charm. In deference, I refrain from matching Papadimitriou's two dozen footnotes. In tribute, however, I adopt his spirit of proud classical reference with something from my own tradition: 18 footnotes, corresponding to the auspicious *gematria* of חי (life).

7. REFERENCES

[1] S. Abiteboul, Z. Abrams, S. Haar, and T. Milo. Diagnosis of asynchronous discrete event systems: Datalog to the rescue! In *PODS*, 2005.

[2] F. N. Afrati and J. D. Ullman. Optimizing joins in a map-reduce environment. In *EDBT*, pages 99–110, 2010.

[3] P. Alvaro, T. Condie, N. Conway, K. Elmeleegy, J. M. Hellerstein, and R. C. Sears. BOOM: Data-centric programming in the datacenter. In *Eurosys*, April 2010.

[4] P. Alvaro, T. Condie, N. Conway, J. M. Hellerstein, and R. Sears. I do declare: Consensus in a logic language. *SIGOPS Oper. Syst. Rev.*, 43(4):25–30, 2010.

[5] P. Alvaro, W. Marczak, N. Conway, J. M. Hellerstein, D. Maier, and R. C. Sears. Dedalus: Datalog in time and space. Technical Report UCB/EECS-2009-173, EECS Department, University of California, Berkeley, Dec 2009.

[6] M. Aref. Datalog for enterprise applications: from industrial applications to research. In *Datalog 2.0 Workshop*, 2010. `http://www.datalog20.org/slides/aref.pdf`.

[7] M. P. Ashley-Rollman, S. C. Goldstein, P. Lee, T. C. Mowry, and P. S. Pillai. Meld: A declarative approach to programming ensembles. In *IEEE International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2007.

[8] A. Atul. Compact implementation of distributed inference algorithms for network. Master's thesis, EECS Department, University of California, Berkeley, Mar 2009.

[9] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[10] J. L. Borges. A new refutation of time. In D. A. Yates and J. E. Irby, editors, *Labyrinths: Selected Stories and Other Writings*. New Directions Publishing, 1964. Translation: James E. Irby.

[11] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and System Sciences*, 25:99–128, 1982.

[12] B. Chandramouli, J. Goldstein, and D. Maier. On-the-fly progress detection in iterative stream queries. In *VLDB*, 2009.

[13] S. Chaudhuri and K. Shim. Query optimization in the presence of foreign functions. In *VLDB*, 1993.

[14] D. Chu and J. Hellerstein. Automating rendezvous and proxy selection in sensor networks. In *Eighth International Conference on Information Processing in*

Sensor Networks (IPSN), 2009.

[15] D. Chu, L. Popa, A. Tavakoli, J. M. Hellerstein, P. Levis, S. Shenker, and I. Stoica. The design and implementation of a declarative sensor network system. In *SenSys*, pages 175–188, 2007.

[16] D. C. Chu. *Building and Optimizing Declarative Networked Systems*. PhD thesis, EECS Department, University of California, Berkeley, Jun 2009.

[17] T. Condie, D. Chu, J. M. Hellerstein, and P. Maniatis. Evita Raced: Metacompilation for declarative networks. *Proc. VLDB Endow.*, 1(1):1153–1165, 2008.

[18] O. Cooper. TelegraphCQ: From streaming database to information crawler. Master's thesis, EECS Department, University of California, Berkeley, 2004.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6), 2007.

[20] N. Durgin, J. C. Mitchell, and D. Pavlovic. A compositional logic for proving security properties of protocols. *Journal of Computer Security*, 11(4):677–721, 2003.

[21] J. Eisner, E. Goldlust, and N. A. Smith. Dyna: a declarative language for implementing dynamic programs. In *Proc. ACL*, 2004.

[22] P. F. Felzenszwalb and D. A. McAllester. The generalized A* architecture. *J. Artif. Intell. Res. (JAIR)*, 29:153–190, 2007.

[23] J. Field, M.-C. V. Marinescu, and C. Stefansen. Reactors: A data-oriented synchronous/asynchronous programming model for distributed applications. *Theor. Comput. Sci.*, 410(2-3), 2009.

[24] E. Fox. *The Five Books of Moses: A New Translation With Introductions, Commentary and Notes*. Schocken Books, 1995.

[25] R. Goldman and J. Widom. WSQ/DSQ: A practical approach for combined querying of databases and the web. In *In SIGMOD*, pages 285–296, 2000.

[26] J. Gray. What next?: A dozen information-technology research goals. *J. ACM*, 50(1):41–57, 2003.

[27] S. Greco and C. Zaniolo. Greedy algorithms in Datalog with choice and negation, 1998.

[28] E. Hajiyev, M. Verbaere, O. de Moor, and K. de Volder. Codequest: querying source code with Datalog. In *OOPSLA*, 2005.

[29] W. Hasan and R. Motwani. Coloring away communication in parallel query optimization. In *VLDB*, 1995.

[30] P. Helland and D. Campbell. Building on quicksand. In *CIDR*, 2009.

[31] J. M. Hellerstein. Toward network data independence. *SIGMOD Rec.*, 32(3):34–40, 2003.

[32] J. M. Hellerstein. Datalog redux: Experience and conjectures. In *PODS*, 2010.

[33] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *SIGMOD*, 1997.

[34] J. M. Hellerstein and M. Stonebraker, editors. *Readings in Database Systems, Third Edition*. Morgan Kaufmann, Mar. 1998.

[35] J. Hennessy and D. Patterson. A conversation with John Hennessy and David Patterson, 2006. `http://queue.acm.org/detail.cfm?id=1189286`.

[36] N. Jain, M. Dahlin, Y. Zhang, D. Kit, P. Mahajan, and P. Yalagandula. STAR: Self-tuning aggregation for scalable monitoring. In *VLDB*, 2007.

[37] T. Jim. SD3: A trust management system with certified evaluation. In *IEEE Symposium on Security and Privacy*, page 106, Washington, DC, USA, 2001. IEEE Computer Society.

[38] H. Karloff, S. Suri, and S. Vassilvitskii. A model of computation for MapReduce. In *Symposium on Discrete Algorithms (SODA)*, 2010.

[39] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Commun. ACM*, 21(7):558–565, 1978.

[40] B. Lampson. Getting computers to understand. *J. ACM*, 50(1):70–72, 2003.

[41] H. C. Lauer and R. M. Needham. On the duality of operating system structures. *SIGOPS Oper. Syst. Rev.*, 13(2):3–19, 1979.

[42] G. Lausen, B. Ludascher, and W. May. On active deductive databases: The Statelog approach. In *In Transactions and Change in Logic Databases*, pages 69–106. Springer-Verlag, 1998.

[43] O. Lhoták and L. Hendren. Jedd: A BDD-based relational extension of Java. In *PLDI*, 2004.

[44] G. M. Lohman. Grammar-like functional rules for representing query optimization alternatives. *SIGMOD Rec.*, 17(3):18–27, 1988.

[45] B. T. Loo. *The Design and Implementation of Declarative Networks*. PhD thesis, University of California, Berkeley, Dec. 2006.

[46] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking. *Commun. ACM*, 52(11):87–95, 2009.

[47] B. T. Loo, T. Condie, M. N. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica. Declarative networking: language, execution and optimization. In *SIGMOD Conference*, pages 97–108, 2006.

[48] B. T. Loo, T. Condie, J. M. Hellerstein, P. Maniatis, T. Roscoe, and I. Stoica. Implementing declarative overlays. In *SOSP*, 2005.

[49] B. T. Loo, J. M. Hellerstein, R. Huebsch, S. Shenker, and I. Stoica. Enhancing P2P file-sharing with an Internet-scale query processor. In *VLDB*, 2004.

[50] B. T. Loo, J. M. Hellerstein, I. Stoica, and R. Ramakrishnan. Declarative routing: extensible routing with declarative queries. In *SIGCOMM*, pages 289–300, 2005.

[51] L. Lu and J. G. Cleary. An operational semantics of Starlog. In *PPDP*, pages 294–310, 1999.

[52] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. TAG: a tiny aggregation service for ad-hoc sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):131–146, 2002.

[53] G. S. Manku, M. Bawa, P. Raghavan, and V. Inc. Symphony: Distributed hashing in a small world. In *In*

Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems, pages 127–140, 2003.

[54] Y. Mao. On the declarativity of declarative networking. *SIGOPS Oper. Syst. Rev.*, 43(4), 2010.

[55] E. Meijer. Confessions of a used programming language salesman. In *OOPSLA*, pages 677–694, 2007.

[56] K. A. Morris. An algorithm for ordering subgoals in NAIL! In *PODS*, 1988.

[57] S. Nath, P. B. Gibbons, S. Seshan, and Z. R. Anderson. Synopsis diffusion for robust aggregation in sensor networks. *TOSN*, 4(2), 2008.

[58] J. A. Navarro and A. Rybalchenko. Operational semantics for declarative networking. In *PADL*, 2009.

[59] O. O'Malley and A. Murthy. Hadoop sorts a petabyte in 16.25 hours and a terabyte in 62 seconds, 2009. `http://developer.yahoo.net/blogs/hadoop/2009/05/hadoop_sorts_a_petabyte_in_162.html`.

[60] C. H. Papadimitriou. Database metatheory: asking the big queries. *SIGACT News*, 26(3), 1995.

[61] A. Perez-Gomez. The myth of Daedalus. *Architectural Association Files*, 10:49–52, 1985.

[62] T. C. Przymusinski. On the Declarative Semantics of Deductive Databases and Logic Programs. In *Foundations of Deductive Databases and Logic Programming*, pages 193–216. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[63] V. Raman, A. Deshpande, and J. M. Hellerstein. Using state modules for adaptive query processing. In *ICDE*, 2003.

[64] K. A. Ross. A syntactic stratification condition using constraints. In *ILPS*, 1994.

[65] T. Urhan and M. J. Franklin. XJoin: A reactively-scheduled pipelines join operator. *Bulletin of the Technical Committee on Data Engineering*, 23(2), 2000.

[66] R. Van Renesse, K. P. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management, and data mining. *ACM Trans. Comput. Syst.*, 21(2):164–206, 2003.

[67] M. Y. Vardi. The complexity of relational query languages (extended abstract). In *STOC*, pages 137–146, 1982.

[68] V. Vianu and D. V. Gucht. Computationally complete relational query languages. In *Encyclopedia of Database Systems*, pages 406–411. Springer, 2009.

[69] J. R. von Behren, J. Condit, F. Zhou, G. C. Necula, and E. A. Brewer. Capriccio: scalable threads for internet services. In *SOSP*, pages 268–281, 2003.

[70] D. Z. Wang, M. J. Franklin, M. Garofalakis, and J. M. Hellerstein. Querying probabilistic declarative information extraction. In *VLDB*, 2010. To appear.

[71] D. Z. Wang, E. Michelakis, M. J. Franklin, M. Garofalakis, and J. M. Hellerstein. Probabilistic declarative information extraction. In *ICDE*, 2010.

[72] M. Welsh, D. Culler, and E. Brewer. SEDA: an architecture for well-conditioned, scalable internet services. *SIGOPS Oper. Syst. Rev.*, 35(5):230–243, 2001.

[73] J. Whaley and M. S. Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. *SIGPLAN Not.*, 39(6):131–144, 2004.

[74] W. White, B. Sowell, J. Gehrke, and A. Demers. Declarative processing for computer games. In *ACM SIGGRAPH Sandbox Symposium*, 2008.

[75] Wikipedia. Eating one's own dog food, 2010. `http://en.wikipedia.org/wiki/Eating_one's_own_dog_food`.

[76] A. N. Wilschut and P. M. G. Apers. Dataflow query execution in a parallel main-memory environment. In *PDIS*, pages 68–77, 1991.

[77] A. Wollrath, G. Wyant, J. Waldo, and S. C. Kendall. A note on distributed computing. Technical Report TR-94-29, Sun Microsystems Laboratories, 1994.

[78] C. Zaniolo and H. Wang. Logic-based user-defined aggregates for the next generation of database systems. In K. Apt, V. Marek, M. Truszczynski, and D. S. Warren, editors, *The Logic Programming Paradigm: Current Trends and Future Directions*. Springer Verlag, 1999.

[79] W. Zhou, Y. Mao, B. T. Loo, and M. Abadi. Unified declarative platform for secure networked information systems. In *ICDE*, 2009.

Querying RDF Streams with C-SPARQL *

Davide Francesco Barbieri Daniele Braga Stefano Ceri
Emanuele Della Valle Michael Grossniklaus
Politecnico di Milano, Dipartimento di Elettronica e Informazione
Piazza L. da Vinci 32, 20133 Milano, Italy
firstname.lastname@elet.polimi.it

ABSTRACT

Continuous SPARQL (C-SPARQL) is a new language for continuous queries over streams of RDF data. C-SPARQL queries consider windows, i.e., the most recent triples of such streams, observed while data is continuously flowing. Supporting streams in RDF format guarantees interoperability and opens up important applications, in which reasoners can deal with knowledge evolving over time. Examples of such application domains include real-time reasoning over sensors, urban computing, and social semantic data. In this paper, we present the C-SPARQL language extensions in terms of both syntax and examples. Finally, we discuss existing applications that already use C-SPARQL and give an outlook on future research opportunities.

1. INTRODUCTION

Stream-based data sources such as sensors, feeds, click streams, and stock quotations have become increasingly important in many application domains. Streaming data are received continuously and in real-time, either implicitly ordered by arrival time, or explicitly associated with timestamps. As it is typically impossible to store a stream in its entirety, Data Stream Management Systems (DSMS) [14], e.g., [19, 12, 4, 1, 3], allow continuously running queries to be registered, that return new results as new data flow within the streams [15]. At the same time, reasoning upon very large RDF data collections is advancing fast, and SPARQL [23] has gained the role of standard query language for RDF data. Also, SPARQL engines are now capable of querying integrated repositories and collecting data from multiple sources. Still, the large knowledge bases now accessible via SPARQL (such as Linked Life Data[1]) are static, and knowledge evolution is not adequately supported.

*This work is supported by the European project LarKC (FP7-215535). Michael Grossniklaus's contribution was carried out under the SNF grant number PBEZ2-121230.

[1]http://www.linkedlifedata.com/

The combination of static RDF data with streaming information leads to **stream reasoning** [13], an important step to enable logical reasoning in real time on huge and noisy data streams in order to support the decision process of large numbers of concurrent users. So far, this step has received little attention by the Semantic Web community. C-SPARQL, that we introduced in [7], is an extension of SPARQL designed to express continuous queries, i.e., queries registered over both RDF repositories and *RDF streams*. C-SPARQL queries can be considered as inputs to specialized reasoners that use their knowledge about a domain to make real-time decisions. In such applications, reasoners operate upon knowledge snapshots, which are continuously refreshed by registered queries. It is important to note that, in this view, reasoners can be unaware of time changes and of the existence of streams. We have also explored the use of reasoners aware of the time-dependent nature of data in [6], where we propose an algorithm for the incremental maintenance of snapshots. Reasoning over streaming facts is also addressed by the authors of [29], who focus on the scalability of reasoning techniques. Another research related to ours is that by Law et al. [17], who put particular emphasis on the problem of mining data streams [18].

In this paper, we present a summary of the description of C-SPARQL published in [7] and apply the language to new use cases. We focus on how C-SPARQL extends SPARQL with functionality required to manage streams, in a way that is comparable to the approach taken by CQL [2]. Note that this paper neither discusses the evaluation and optimization of C-SPARQL queries, nor other entailment regimes beyond basic RDF entailment. Details on how we addressed these topics in the context of C-SPARQL can be found in [6, 7].

Bolles et al. [10] presented a first attempt to extend SPARQL to support streams, that can be considered an antecedent of our work. It introduced

a syntax for the specification of logical and physical windows in SPARQL queries by means of local grammar extensions. However, their approach is different from ours at least in two key aspects. First, they simply introduce RDF streams as a new data type, and omit essential ingredients, such as aggregates and timestamp functions. Second, the authors do not follow the established approach where windows are used to transform streaming data into non-streaming data in order to apply standard algebraic operations. Instead, they chose to change the standard SPARQL operators by making them timestamp-aware and, thereby, actually introduce a new language semantics.

In stream processing, aggregation is an important functionality. When we started working on C-SPARQL, we based it on SPARQL 1.0 which does not contain any support for aggregates. In previous publications, we therefore also introduced our own syntax and semantics for aggregates in C-SPARQL that does not shrink results in the presence of grouping [7]. In the meantime, it is foreseeable that the upcoming SPARQL 1.1 specification will include aggregation functionality similar to the one known from SQL. For this paper and future work on C-SPARQL, we have chosen to align our notion of aggregates with the one proposed by the W3C and present all examples accordingly.

Furthermore, several SPARQL implementations support some form of proprietary aggregation functions and group definitions. OpenLink Virtuoso[2] supports COUNT, COUNT DISTINCT, MAX, MIN and AVG, with implicit grouping criteria. ARQ[3] supports COUNT and COUNT DISTINCT over groups defined through an SQL-like GROUP BY clause. ARC[4] also supports the keyword AS to bind variables to aggregated results. In [25], the authors study how grouping and aggregation can be defined in the context of queries over RDF graphs, taking into consideration the peculiarities of the data model, and providing an extension of SPARQL based on operational semantics.

This paper is organized as follows. Section 2 presents the distinguishing language extensions of C-SPARQL referring to a simple scenario of social data analysis. After introducing the RDF stream data type, we discuss the extensions for windows, stream registration, and query registration. Other application scenarios, beyond social data analysis, are presented in Section 3. Finally, an outlook on using C-SPARQL for enabling stream reasoning is presented in Section 4.

[2]http://virtuoso.openlinksw.com/
[3]http://jena.sourceforge.net/ARQ/
[4]http://arc.semsol.org/

2. C-SPARQL

In the following, we present a summary of C-SPARQL by progressively introducing its new features relative to SPARQL. We interleave the presentation of the new syntax, extended by adding new productions to the standard grammar of SPARQL [23], and the discussion of some examples. As a demonstration scenario, we have chosen queries that are relevant to a (highly simplified) case of social data analysis.

2.1 RDF Stream Data Type

C-SPARQL adds **RDF streams** to the SPARQL data types, in the form of an extension done much in the same way in which the stream type has been introduced to extend relations in relational data stream management systems. RDF streams are defined as ordered sequences of pairs, each pair being made of an RDF triple and a timestamp τ:

$$\begin{array}{c}\ldots\\(\langle subj_i, pred_i, obj_i\rangle\ ,\ \tau_i)\\(\langle subj_{i+1}, pred_{i+1}, obj_{i+1}\rangle\ ,\ \tau_{i+1})\\\ldots\end{array}$$

Timestamps can be considered as *annotations* of RDF triples, and are monotonically non-decreasing in the stream ($\tau_i \leq \tau_{i+1}$). More precisely, timestamps are not strictly increasing because they are not required to be unique. Any (unbounded, though finite) number of consecutive triples can have the same timestamp, meaning that they "occur" at the same time, although sequenced in the stream according to a positional order.

Example. The classes and properties that we consider in the social data analysis scenario are described in the schema of Figure 1. All class instances are identified by URLs.

Users also have names, and, by virtue of two properties, they *know* and *follow* other users. Using well-known Semantic Web vocabularies, the user name and the `foaf:knows` property can be described using the Friend of a Friend vocabulary (FOAF) [9]. For the `sioc:follows` property, we can use Semantically-Interlinked Online Communities (SIOC) [8].

Topics represent entities of the real world (such as movies or books, to give examples that are relevant for our scenario), with a name and a category.

Documents represent information sources on actual topics. Examples of documents are Web pages that *describe* topics like a particular book or movie. As vocabularies, we can use `rdfs:label` [11] for the names of documents and topics. Finally, the attribute `skos:subject` from the Simple Knowledge Organization System (SKOS) [22] connects a topic to its category, identified using YAGO [27].

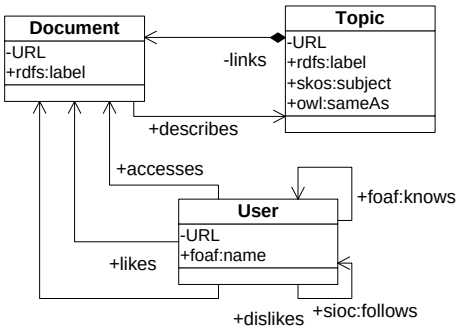


Figure 1: Example data schema

All the knowledge described so far is static (or, more precisely, slowly changing), meaning that the applications we are willing to consider can assume this information as invariant in a period comparable with the size of a window. Of course, updates of this information are also allowed, e.g., to state that a new friendship holds after the addition of an instance of the `foaf:knows` property.

The running example also uses streaming knowledge, and namely streams of notifications that capture the behavior of users with respect to documents (and therefore, transitively, to topics). The *accesses*, *likes*, and *dislikes* properties represent events that occur at the time in which users access a document or express their opinion about it.[5] Quite straightforwardly, any interaction of a user with a document generates in the stream a triple of the form $\langle U, sd{:}accesses, D\rangle$, where U and D respectively represent a generic user and a generic document. Also, selected interactions of users with documents generate triples of the form $\langle U, sd{:}likes, D\rangle$ and $\langle U, sd{:}dislikes, D\rangle$. It is worth noting that in the stream the predicates can only assume one of the three values exemplified above, while the subjects and objects may freely vary in the space of users and documents. This is coherent with RDF repositories whose predicates are taken from a small vocabulary constituting a sort of schema. However, the interpretation of C-SPARQL makes no specific assumptions nor requires restrictions on variable bindings relative to any part of the streaming triples. An example of possible triples in a stream of interactions and opinions is given below.

triple	Timestamp
`c:Usr1 sd:accesses c:movie1`	t_{400}
`c:Usr2 sd:accesses c:movie1`	t_{401}
`c:Usr1 sd:likes c:movie2`	t_{402}
...	...

[5]In the rest of this paper, we refer to this vocabulary with the prefix `sd` (for "social data").

2.2 Windows

The introduction of data streams in C-SPARQL requires the ability to *identify* such data sources and to specify *selection* criteria over them.
For *identification*, we assume that each data stream is associated with a distinct IRI, that is a locator of the actual data source of the stream. More specifically, the IRI represents an IP address and a port for accessing streaming data.
For *selection*, given that streams are intrinsically infinite, we introduce a notion of windows on streams, whose types and characteristics are inspired by the ones defined for relational streaming data.

Identification and selection are expressed in C-SPARQL by means of the `FROM STREAM` clause. The syntax is as follows:

FromStrClause ⟶ 'FROM' ['NAMED'] 'STREAM' *StreamIRI* '[RANGE' *Window* ']'
Window ⟶ *LogicalWindow* | *PhysicalWindow*
LogicalWindow ⟶ *Number TimeUnit WindowOverlap*
TimeUnit ⟶ 'ms' | 's' | 'm' | 'h' | 'd'
WindowOverlap ⟶ 'STEP' *Number TimeUnit* | 'TUMBLING'
PhysicalWindow ⟶ 'TRIPLES' *Number*

A window extracts the *last* data elements from the stream, which are the only part of the stream to be considered by one execution of the query. The extraction can be *physical* (a given number of triples) or *logical* (all triples occurring within a given time interval, whose number is variable over time).

Logical windows are *sliding* [16] if they are progressively advanced by a given `STEP` (i.e., a time interval that is shorter than the window's time interval). They are *non-overlapping* (or `TUMBLING`) if they are advanced in each iteration by a time interval equal to their length. With tumbling windows every triple of the stream is included exactly into one window, whereas with sliding windows some triples can be included into several windows.

The optional `NAMED` keyword works exactly like when applied to the standard SPARQL `FROM` clause for tracking the provenance of triples. It binds the IRI of a stream to a variable which is later accessible through the `GRAPH` clause.

Example. As a very simple first example, consider the query that extracts all books (i.e., all topics whose category is "book") seen by the friends of John in the last 15 minutes. The query considers the last 15 minutes, and the sliding window is modified every minute, so that the query result is renewed every minute.

```
SELECT DISTINCT ?topic
FROM STREAM <http://streamingsocialdata.org/
             interact.trdf> [RANGE 15m STEP 1m]
WHERE { ?user sd:accesses ?document .
        ?user foaf:knows ?john .
        ?john foaf:name "John" .
        ?document t:describes ?topic .
        ?topic skos:subject yago:Movies . }
```

The query joins static and streaming knowledge, and is executed as follows. First, all triples with `sd:accesses` as a predicate are extracted from the current window over the stream, to match the first triple pattern in the WHERE clause. Then the other triple patterns are matched against the static knowledge base, applying the "join" conditions expressed by the bindings of variables `?user` and `?document` to identify the observed `?topic`s. The window considers all the stream triples in the last 15 minutes, and is advanced every minute. This means that at every new minute new triples enter into the window and old triples exit from the window. Note that the query result does not change during the slide interval, and is only updated at every slide change. Triples arriving in the stream between these points in time are queued until the next slide change and do not contribute to the result until then.

2.3 Stream Registration

The result of a C-SPARQL query can be a set of bindings, but also a new RDF stream. In order to generate a stream, the query must be registered through the following statement:

Registration ⟶ 'REGISTER STREAM' *QueryName*
['COMPUTED EVERY' *Number TimeUnit*] 'AS' *Query*

Only queries in the CONSTRUCT and DESCRIBE form[6] can be registered as generators of RDF streams, as they produce RDF triples, associated with a timestamp as an effect of the query execution.

The optional COMPUTED EVERY clause indicates the frequency at which the query *should* be computed. If no frequency is specified, the query is computed at a frequency that is automatically determined by the system.[7]

[6] There are four query forms in SPARQL, different in the first clause: SELECT returns variables bound in a query pattern match. CONSTRUCT returns an RDF graph constructed by substituting variables in a set of triple templates. ASK returns a boolean indicating whether a query pattern matches or not. DESCRIBE returns an RDF graph that describes the resources found. Please refer to [23] for further explanations.

[7] Several data stream management systems are capable of self tuning the execution frequency of registered queries. This not only applies to queries with unspecified registration frequencies, but also whenever, due to peaks of workload, the execution frequency of all queries is reduced, so as to gracefully degrade the overall performances.

Example. The following example shows the construction of a new RDF data stream by means of the registration of a CONSTRUCT query. We consider the previous example again, and modify it so as to generate a stream by selecting all interactions that are of the "likes" type, that are performed by a friend of John, and that concern movies.

```
REGISTER STREAM MoviesJohnsFriendsLike
        COMPUTED EVERY 5m AS
CONSTRUCT {?user sd:likes ?document}
FROM STREAM <http://streamingsocialdata.org/
             interact.trdf> [RANGE 30m STEP 5m]
WHERE { ?user sd:likes ?document .
        ?user foaf:knows ?john .
        ?john foat:name "John" .
        ?document sd:describes ?topic .
        ?topic skos:subject yago:Movies . }
```

This query uses the same logical conditions as the previous one on static data, but only matches the `sd:likes` predicate. The output is constructed in the format of a stream of RDF triples. Every query execution may produce from a minimum of zero triples to a maximum of an entire graph. The timestamp is always dependent on the query execution time only, and is not taken from the triples that match the patters in the WHERE clause. Thus, even though in the example the output stream is a restriction of the input stream, a new timestamp is assigned to every triple. Also note that, if the window contains more than one matching triple with a `sd:likes` predicate, then also the result contains more than one triple, that are returned as a graph. In this case the same timestamp is assigned to all the triples of the graph. In all cases, however, timestamps are system-generated in monotonic non-decreasing order. Results of two evaluations of the previous query are presented in the table below, generating two graphs (one at $\tau = 100$ and one at $\tau = 101$).

triple	Timestamp
`c:Usr1 sd:likes c:Movie1`	t_{100}
`c:Usr2 sd:likes c:Movie2`	t_{100}
`c:Usr1 sd:likes c:Movie2`	t_{101}
`c:Usr2 sd:likes c:Movie1`	t_{101}
`c:Usr3 sd:likes c:Movie3`	t_{101}

2.4 Query Registration

All queries over RDF data streams are denoted as *continuous queries*, because they continuously produce output in the form of tables of variable bindings or RDF graphs. In the section above we addressed the registration of RDF streams. Here, we address the registration of queries that do not produce a stream, but a result that is periodically updated. C-SPARQL queries are registered through the following statement:

Registration ⟶ 'REGISTER QUERY' *QueryName*
['COMPUTED EVERY' *Number TimeUnit*] 'AS' *Query*

The COMPUTED EVERY clause is the same as the one for stream registration.

Example. As a very simple example of a registered query that does not generate a stream, consider the following query. For each known user, the query counts the overall number of interactions performed in the last 30 minutes and the number of distinct topics to which the documents refer.

```
REGISTER QUERY GlobalCountOfInteractions
         COMPUTED EVERY 5m AS
SELECT ?user
       COUNT(?document) as ?numberOfInteractions
       COUNT(DISTINCT ?topic) as ?numDifferentTopics
FROM STREAM <http://streamingsocialdata.org/
             interact.trdf> [RANGE 30m STEP 5m]
WHERE { ?user sd:accesses ?document .
        ?document sd:describes ?topic . }
GROUP BY { ?user }
```

The query is executed by matching all interactions in the window, grouping them by ?user, and computing the aggregates. The result has the form of a table of bindings that is updated every 5 minutes.

All the examples considered so far have shown a join of static and streaming knowledge. As an example of query composability, we now show a query that takes as input the registered stream generated by the query shown in Section 2.3.

```
REGISTER QUERY GlobalCountOfInteractions
         COMPUTED EVERY 5m AS
SELECT ?user COUNT(?document) as ?numberOfMovies
FROM STREAM <http://streamingsocialdata.org/
     MoviesJohnsFriendsLike.trdf> [RANGE 30m STEP 5m]
WHERE { ?user sd:likes ?document }
GROUP BY { ?user }
```

The query counts, among the friends of John, the number of movies that each friend has liked in the last 30 minutes.

2.5 Multiple Streams

C-SPARQL queries can combine triples from multiple RDF streams, as shown in the following example.

Example. In addition to the stream of interactions, we now consider the presence of a second stream of data concerning the entrance of registered users into theaters to watch movies. The next query takes as input the stream of preferences of John's friends and the stream about people entering cinemas, and identifies friends who like a 3D movies, but only considering users who actually watched at least two 3D movies in the last week (so as to focus on the advice of "experts").

```
REGISTER QUERY JohnsFriendsToRecommend3DMovies AS
SELECT ?user
FROM STREAM <http://streamingsocialdata.org/
          MoviesJohnsFriendsLike.trdf> [RANGE 1h]
FROM STREAM <http://comingsoon.com/
       WatchedMovies.trdf>  [RANGE 7d]
WHERE { ?user sd:likes ?document .
        ?document sd:describes ?topic .
        ?topic skos:subject yago:3DMovies .
        { SELECT ?user
          WHERE { ?user sd:accesses ?document1 .
                  ?document1 sd:describes ?topic1 .
            ?topic1 skos:subject yago:3DMovies . }
          GROUP BY ?user
          HAVING COUNT(DISTINCT ?topic1) >= 2 } }
```

The query is executed as follows. Variable ?user is matched in the WHERE clause of the outer query among the friends of John. Also, the topic is checked to be a 3D movie (the stream is selected checking that the topics are classified as generic movies). The user is also checked to have the property of having seen at least two other 3D movies in the nested query. Note the use of the same ?user variable in the nested query so as to pass the binding and check the "aggregate" property.

2.6 Timestamp Function

The timestamp of a stream element can be retrieved and bound to a variable using a timestamp function. The timestamp function has two arguments.

- The first is the name of a variable, introduced in the WHERE clause and bound to an RDF triple of that stream by pattern matching.
- The second (optional) is the URI of a stream, that can be obtained through SPARQL GRAPH clause.

The function returns the timestamp of the RDF stream element producing the binding. If the variable is not bound, the function is undefined, and any comparison involving its evaluation has a non-determined behavior. If the variable gets bound multiple times, the function returns the most recent timestamp value relative to the query evaluation time.

Example. In order to exemplify the use of timestamps within queries, we show a query that tries to discover causal relationships between different actions. More precisely, the query identifies users who are likely to influence the behavior of other users, by matching interactions of the same kind that occur on the same document *after* the first user has performed them. The query in C-SPARQL is the following:

```
REGISTER STREAM OpinionMakers
         COMPUTED EVERY 5m AS
SELECT ?opinionMaker
FROM STREAM <http://streamingsocialdata.org/
     interact.trdf>  [RANGE 30m STEP 5m]
```

```
WHERE { ?opinionMaker foaf:knows ?friend .
        ?friend ?opinion ?document.
        ?opinionMaker ?opinion ?document .
        FILTER ( timestamp(?friend) >
                 timestamp(?opinionMaker)
                   && ?opinion != sd:accesses ) }
GROUP BY ( ?opinionMaker )
HAVING ( COUNT(DISTINCT ?friend) > 3 )
```

Note that the timestamps are taken from variables that occur only once in patterns applied to streaming triples, thus avoiding ambiguity. Also, the query filters out actions of type "accesses", that are normally required before expressing an opinion such as "like" or "dislike".

3. APPLICATIONS

The scenario of social data analysis is just one example of many possible applications of C-SPARQL. In the last years, more and more effort has been put in trying to address problems that require reasoning on streaming data, and this has been done mainly with "classical" reasoning tools. For instance, Bandini et al. [5] worked on traffic monitoring and traffic pattern detection. Mendler et al. [21] applied constructive Description Logics to financial-transaction auditing. In the mobile telecommunication sector, Luther et al. [20] reported the need for reasoning over streams for situation-aware mobile services. Walavalkar et al. [29] worked on patient monitoring systems. All these application areas are natural settings for C-SPARQL. In the following, we provide more details about concrete applications of C-SPARQL in the cases of situation aware mobility and oil production. In Section 4, we will also outline how we are currently studying dedicated reasoning techniques for the interplay of C-SPARQL and reasoners, in order to efficiently carry out reasoning tasks over streams.

3.1 Situation-Aware Mobility

Mobility is one of the defining characteristics of modern life. Technology can support and accompany mobility in several ways, both for business and for pleasure. Mobile phones provide a good basis for challenging C-SPARQL use cases, as they are popular and widespread. In order to complete the adoption of such devices in our everyday life, mobile applications must fulfill real-time requirements, especially if we are to use them to make short-term decisions. Leveraging data from sensors, which is likely to be available in the form of streams, mobile applications may compute interesting answers by reasoning over streams.

The following C-SPARQL query finds the locations of commuters having less than 30 minutes of travel time remaining. For each user, it retrieves the train number, its position in terms of the closest station, the city where the station is in, etc., by computing the transitive closure of relation `isIn`.

```
REGISTER QUERY WhereAlmomstToDestinationCommutersAre
        COMPUTED EVERY 1sec AS
SELECT DISTINCT ?user ?location
FROM <http://mobileservice.org/meansOfTransp.rdf>
FROM STREAM <http://mobileservice.org/
             positions.trdf> [RANGE 10sec STEP 1sec]
WHERE { ?user ex:isIn ?location .
        ?user a ex:Commuter .
        ?user ex:remainingTravelTime ?t .
        FILTER ( ?t >= "PT30M"^^xsd:duration )  }
```

It does so by continuously querying a stream of RDF triples that describe the users on trains, moving from a station to another, together with a static RDF graph, which describes where the stations are located, e.g., a station is in a city, which is in a region, which is in a state, etc. For further information about this application scenario, the reader is directed to [6].

3.2 Oil Production

Oil operation engineers base their decision process on real time data acquired from sensors on oil rigs, located at the sea surface and seabed. A typical oil production platform is equipped with about 400.000 sensors for measuring environmental and technical parameters. The problems they face include determining the expected time to failure whenever the barring starts vibrating, given the brand of the turbine, or detecting weather events from observation data. For details about this application scenario, the reader is directed to [26].

The C-SPARQL query below detects if a weather station is observing a blizzard. A blizzard is identified when a severe storm, characterized by low temperatures, strong winds, and heavy snow, lasts for 3 hours or more.

```
REGISTER STREAM BlizzardDetection
        COMPUTED EVERY 10m AS
CONSTRUCT {?s so:generatedObservation [a w:blizzard]}
FROM <http://oilprod.org/weatherStations.rdf>
FROM STREAM <http://oilprod.org/weatherObs.trdf>
  [RANGE 3h STEP 10m]
WHERE {
 ?s grs:point "66.348085,10.180662" ;
  so:generatedObservation [a w:SnowfallObservation] .
 { SELECT ?s
   WHERE { ?s so:generatedObservation ?o1
           ?o1 a w:TemperatureObservation ;
             so:observedProperty w:AirTemperature ;
             so:result [ so:value ?temperature  ] . }
   GROUP BY ( ?s )
   HAVING (AVG(?temperature)<"0.0"^^xsd:float) }
 { SELECT ?s
   WHERE { ?s so:generatedObservation ?o2
           ?o2 a w:WindObservation ;
             so:observedProperty w:WindSpeed ;
             so:result [ so:value ?speed  ] . }
   GROUP BY ( ?s )
   HAVING (MIN(?speed)> "40.0"^^xsd:float)  }
}
```

4. OUTLOOK

We believe that C-SPARQL and its corresponding infrastructure provide an excellent starting point for stream reasoning [13]. By providing an RDF-based representation of heterogeneous streams, C-SPARQL solves the challenge of giving reasoners an access protocol for heterogeneous streams. As RDF is the most accepted format to feed information to reasoners, C-SPARQL allows existing reasoning mechanisms to be further extended in order to support continuous reasoning over data streams and rich background knowledge. We already made a first step in this direction, investigating the incremental maintenance of ontological entailment materializations [6]. To do so, we annotate streaming knowledge with expiration times, which we manage in an auxiliary data structure, devoted to handle the limited validity of inference through time. Our reasoner is then capable of incrementally maintaining the entailments of transient knowledge, that are themselves transient, in an efficient way. In future work, we plan to extend this approach and to generalize it to more expressive languages.

Moreover, the extraction of patterns from data streams is subject of ongoing research in machine learning. For instance, results from statistical relational learning are able to derive classification rules from example data in very effective ways. In our future work, we intend to link relational learning methods with C-SPARQL to facilitate pattern extraction on top of RDF streams.

Finally, we envision the possibility to leverage recent developments in distributed and parallel reasoning [28, 24] for scaling up to large data streams and many concurrent reasoning tasks.

5. REFERENCES

[1] D. J. Abadi et al. The Design of the Borealis Stream Processing Engine. In *Proc. CIDR*, 2005.

[2] A. Arasu, S. Babu, and J. Widom. The CQL Continuous Query Language: Semantic Foundations and Query Execution. *The VLDB Journal*, 15(2):121–142, 2006.

[3] Y. Bai, H. Thakkar, H. Wang, C. Luo, and C. Zaniolo. A Data Stream Language and System Designed for Power and Extensibility. In *Proc. CIKM*, 2006.

[4] H. Balakrishnan et al. Retrospective on Aurora. *The VLDB Journal*, 13(4):370–383, 2004.

[5] S. Bandini, A. Mosca, and M. Palmonari. Common-sense spatial reasoning for information correlation in pervasive computing. *Applied Artificial Intelligence*, 21(4&5):405–425, 2007.

[6] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Incremental Reasoning on Streams and Rich Background Knowledge. In *ESWC*, 2010.

[7] D. F. Barbieri, D. Braga, S. Ceri, and M. Grossniklaus. An Execution Environment for C-SPARQL Queries. In *Proc. EDBT*, 2010.

[8] U. Bojars, J. G. Breslin, A. Finn, and S. Decker. Using the semantic web for linking and reusing data across web 2.0 communities. *Web Semantics*, 6(1):21–28, 2008.

[9] U. Bojars, J. G. Breslin, V. Peristeras, G. Tummarello, and S. Decker. Interlinking the social web with semantics. *Intelligent Systems*, 23(3):29–40, 2008.

[10] A. Bolles, M. Grawunder, and J. Jacobi. Streaming SPARQL – Extending SPARQL to Process Data Streams. In *Proc. ESWC*, 2008.

[11] D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, W3C Working Draft. Technical report, W3C, 2002.

[12] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In *Proc. SIGMOD*, 2000.

[13] E. Della Valle, S. Ceri, F. van Harmelen, and D. Fensel. It's a Streaming World! Reasoning upon Rapidly Changing Information. *IEEE Intelligent Systems*, 24(6):83–89, 2009.

[14] M. Garofalakis, J. Gehrke, and R. Rastogi. *Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., 2007.

[15] L. Golab, D. DeHaan, E. D. Demaine, A. López-Ortiz, and J. I. Munro. Identifying Frequent Items in Sliding Windows over On-line Packet Streams. In *IMC*, 2003.

[16] L. Golab and M. T. Özsu. Processing Sliding Window Multi-Joins in Continuous Queries over Data Streams. In *Proc. VLDB*, 2003.

[17] Y.-N. Law, H. Wang, and C. Zaniolo. Query Languages and Data Models for Database Sequences and Data Streams. In *Proc. VLDB*, 2004.

[18] Y.-N. Law and C. Zaniolo. An Adaptive Nearest Neighbor Classification Algorithm for Data Streams. In *Proc. PKDD*, 2005.

[19] L. Liu, C. Pu, and W. Tang. Continual Queries for Internet Scale Event-Driven Information Delivery. *IEEE Trans. Knowl. Data Eng.*, 11(4):610–628, 1999.

[20] M. Luther, Y. Fukazawa, M. Wagner, and S. Kurakake. Situational reasoning for task-oriented mobile service recommendation. *Knowledge Eng. Review*, 23(1):7–19, 2008.

[21] M. Mendler and S. Scheele. Exponential Speedup in UL Subsumption Checking relative to general TBoxes for the Constructive Semantics. In *Proc. DL*, 2009.

[22] A. Miles, B. Matthews, M. Wilson, and D. Brickley. SKOS core: Simple Knowledge Organisation for the Web. In *Proc. Intl. Conf. on Dublin Core and metadata applications*, Madrid, Spanien, 2005.

[23] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. `http://www.w3.org/TR/rdf-sparql-query/`.

[24] A. Schlicht and H. Stuckenschmidt. Distributed resolution for expressive ontology networks. In *Web Reasoning and Rule Systems*, 2009.

[25] D. Y. Seid and S. Mehrotra. Grouping and Aggregate queries Over Semantic Web Databases. In *Proc. Intl. Conf. on Semantic Computing (ICSC)*, 2007.

[26] H. Stuckenschmidt, S. Ceri, E. Della Valle, and F. van Harmelen. Towards expressive stream reasoning. In *Proceedings of the Dagstuhl Seminar on Semantic Aspects of Sensor Networks*, 2010.

[27] F. M. Suchanek, G. Kasneci, and G. Weikum. YAGO: a core of semantic knowledge unifying wordnet and wikipedia. In *Proc. WWW*, 2007.

[28] J. Urbani, S. Kotoulas, J. Maassen, F. van Harmelen, and H. Bal. OWL reasoning with WebPIE: calculating the closure of 100 billion triples. In *Proc. ESWC*, 2010.

[29] O. Walavalkar, A. Joshi, T. Finin, and Y. Yesha. Streaming Knowledge Bases. In *Proc. SSWS*, 2008.

Beyond Isolation: Research Opportunities in Declarative Data-Driven Coordination

Lucja Kot, Nitin Gupta, Sudip Roy, Johannes Gehrke, and Christoph Koch

Cornell University
Ithaca, NY 14853, USA
{lucja, niting, sudip, johannes, koch}@cs.cornell.edu

ABSTRACT

There are many database applications that require users to coordinate and communicate. Friends want to coordinate travel plans, students want to jointly enroll in the same set of courses, and busy professionals want to coordinate their schedules. These tasks are difficult to program using existing abstractions provided by database systems because in addition to the traditional ACID properties provided by the system they all require some type of *coordination* between users. This is fundamentally incompatible with isolation in the classical ACID properties.

In this position paper, we argue that it is time for the database community to look beyond isolation towards principled and elegant abstractions that allow for communication and coordination between some notion of (suitably generalized) transactions. This new area of *declarative data-driven coordination* (D3C) is motivated by many novel applications and is full of challenging research problems. We survey existing abstractions in database systems and explain why they are insufficient for D3C, and we outline a plethora of exciting research problems.

1. INTRODUCTION

Every sin is the result of collaboration. — Stephen Crane

1.1 Databases and Social Applications

In his book "The Seven Habits of Highly Effective People," Stephen Covey outlines seven inspirational habits that lead the reader on a path to maturing from dependence to independence and finally to interdependence. As people become interdependent, they start working together to achieve a common goal through coordination and collaboration. Do databases need to go a similar path and move beyond independence?

In the last decade, the amount of data on the Internet has grown at a staggering rate. The rise of Web 2.0 has fueled that growth to no small extent by providing platforms that enable people to create, upload, and share content very easily. However, people do much more than just produce and consume data; they are beginning to center their life around the Web and use social applications for very complex data-driven tasks.

As a concrete, seemingly mundane example of a social application, consider a simple travel booking application. Our community is very familiar with these as they have involved databases for many years. Yet all of us have faced the scenario where we wished to not just book travel tickets for ourselves, but to *coordinate* travel plans with family, friends or colleagues. How does this coordination happen today? Typically, it starts with a lengthy phone or email conversation to decide on the itinerary. Then, one person books tickets for everyone, and there is another round of emails to sort out the finances. Or everybody arranges to book at about the same time, hoping that the airline seats do not fill up in the meanwhile.

Coordinating travel plans manually is not just inefficient, it is unsatisfying from a system design standpoint. Certainly it is possible to build the application that handles this travel booking scenario; we just need to find some guys in a garage to code it up perhaps using triggers, nested transactions or special-purpose application code and data structures. (We will discuss these potential implementations shortly.) However, it is worrisome that such a simple use case like cooperative travel booking, which has motivated so much database research (and database exam questions), today still requires ad-hoc "hacks" for this conceptually simple and useful functionality.

The fact that modern databases are difficult to use in collaborative settings is not an accident. The requirements from the social application of coordinating some actions clashes with a fundamental abstraction that is a cornerstone of the database community: the ACID transaction. Transactions were designed to be isolated from each other; therefore, the transaction abstraction provides no mechanism for coordination. In social applications, transactions are still a meaningful concept, and atomicity and durability are still crucial. Isolation, however, may be too strict a requirement and in the use case above has become an obstacle that the application programmer has to navigate around.

Coordinating transactions would certainly facilitate basic

coordination with a small group of friends on travel bookings, leisure activities and course enrollments [5]. However, there are many other applications that would benefit from various types of transaction coordination.

For example, consider a professor who wishes to schedule a weekly meeting with each advisee, ensuring that everyone's availability is respected and that no two meetings fall in the same slot. More generally, people frequently want to coordinate individual contributions to meet a given goal. Some simple examples are wedding gift purchases or potluck dinner planning to ensure a balanced meal. In a more serious vein, grass-roots groups frequently want to organize coordinated events, whether in the aftermath of a disaster, for social or environmental activism, or other reasons. Some social networking sites today already enable a limited form of volunteer coordination, but clearly much more could be achieved with a suitable technological solution.

Sometimes, users want to agree on more than just binary yes/no decisions relating to participation or contributions. Charities run fundraising drives which include gift matching pledges to encourage more donations. If multiple charities are involved in a single drive, the donation coordination problem can become quite complex [2].

Support for complex coordination would also be very useful in massively multiplayer online games. These games typically involve thousands of players, most of them unknown to each other. Groups of players often want to work together to achieve certain goals. A mechanism allowing users to communicate and coordinate plans of action with partners would significantly enhance the gameplay experience, particularly if it allowed partnerships to be formed on-the-fly based on shared player goals.

It is clear that data-driven coordination is applicable to a wide variety of domains and can take many forms. However, to date, coordination has not been recognized as fundamental to a wide range of data-intensive applications, and has only been implemented using ad-hoc solutions.

1.2 Life Beyond Isolation

We believe that it is time to change this pitiful situation. We need research into *declarative data-driven coordination* (D3C), abstractions that extend transactions beyond isolation to allow some form of information flow. We postulate that the fundamental design principle behind D3C should be that in the data-driven setting, coordination should be expressed in a *declarative* way. That is, programmers should only specify *what* coordination is needed, not *how* it should be achieved. Declarativity has long been an underlying design principle in databases, and we believe that it is as applicable to coordination as it is to querying. The complexity of dealing with details of concurrency and coordination should be moved away from the programmer and into the system, where it belongs.

In a system designed to support D3C, our travel booking coordination scenario might play out as follows. Suppose a user, Catherine, wants to travel to Paris on the same flight as her friend Sylvia. Rather than communicating out-of-band via email or phone, Catherine would simply specify her database update as follows, in an SQL-like language or in a suitable visual interface:

```
COMMUNICATE flightno IN
INSERT INTO booking
SELECT 'Catherine', flightno
FROM flights
WHERE destination = 'Paris'
COORDINATING WITH
SELECT 'Sylvia', flightno
FROM flights
WHERE destination = 'Paris'
```

Assuming Sylvia wants to coordinate with Catherine, she would issue a symmetric update with the strings 'Catherine' and 'Sylvia' exchanged. Each statement asks to book a seat on a single but arbitrary flight to Paris on which the other friend also books a flight. However, this constraint across updates is not equivalent to a WHERE-clause, which in this case would never allow either friend to book a flight. Instead, coordination on flightno works like a postcondition on the updates: the updates are only allowed to fire if the execution of both updates will cause the coordination constraint to be satisfied right after the updates.

The system would recognize that the two transactions want to coordinate, and make bookings for each user accordingly. The transactions would remain separate and isolated except for the single information exchange about the flight number. With multiple flights to choose from and no further constraints given, there is a degree of nondeterminism here; the system has to choose one suitable flight and make reservations for both friends.

The principle of coordination by queries and updates just sketched raises many research questions. The goal of the remainder of this article is to voice many of them, and to outline a research program with the aim of achieving the vision of declarative data-driven coordination.

2. D3C AND MODERN DATABASES

Although isolation has always been fundamental to transactions, database research has taken some steps towards enabling coordination over the years. A few abstractions allow a limited form of information exchange among transactions. In addition, some database mechanisms can be used to implement forms of coordination, although they were not originally designed for this purpose.

The best-known abstraction that allows some form of communcation between transactions is nested transactions [7]. In nested transactions, inner transactions can pass information to the outer transaction using variables. However, this model is very restricted in that it does not permit bidirectional information flow between inner transactions. The type of coordination shown in our previous example – Cather-

ine and Sylvia's flight booking – requires mutual information exchange in both directions: Catherine needs to select the same flight number as Sylvia, and both these decisions need to made at the same time as otherwise there may be no longer a seat available after booking the first seat. Thus nested transactions unfortunately cannot be used to implement coordination.

Triggers are an alternate mechanism that might be used to implement coordination [13]. However, triggers were not originally designed for this purpose, and therefore fall short in many ways as a solution for D3C. First, like nested transactions, they do not easily enable bidirectional information flow between transactions. Second, triggers are designed to *initiate* transactions. No related constructs or abstractions exist to "pause" transactions while their progress depends on some other activity in the system, as would be needed in many of our D3C scenarios. Finally, triggers are notoriousy difficult to manage if orderly and controlled behavior is needed. Maintaining transactional properties like atomicity and durability as well as handling the consequences of decisions about committing and aborting would become highly complex in a trigger-based system.

Since coordination is currently not supported at the database level, today's developers must implement it in application code. Such an implementation must include all the coordination mechanisms, which are nontrivial to design and program. These mechanisms need to make heavy use of the underlying database as they involve database state, and they themselves need to persist data in case a coordination needs to wait to take place in the future. Thus implementing coordination at the application level creates a tight coupling between middle and database tier, a suboptimal design. In addition, implementing coordination at the middle tier may remove opportunities for optimization of the coordination within the database.

Coordination Beyond Databases. Our community's thinking which is solely centered on isolation is in contrast with other fields of computer science, which have long permitted, supported, and made good use of communication between concurrent tasks. For example, operating systems typically provide low-level mechanisms such as message passing, shared memory, locks, and semaphores that enable programs to coordinate their execution. There are also efficient higher-level models of coordination between programs, such as transactional memory [6].

Formalisms like the Pi-calculus [8] provide abstractions such as *channels* between processes; these abstractions allow to model and reason about the execution of communicating programs. Channels and other communication methods are also available to programmers as implementation mechanisms. Many programming languages already come with concurrency support – to name only a few, Concurrent ML [9], Erlang [11], Stackless Python [1] and Concurrent Haskell [4]. Cmmunication also plays an important role in multiagent systems [12] where it facilitates tasks such as distributed planning and decision making.

For many years, the database community was able to get away with a comparative lack of attention to communication. Most traditional applications did not require it, and assuming full isolation allowed for design simplifications and optimizations. However, modern collaborative data-driven tasks expose the limitations of full isolation among transactions. As a community, it is time for us to expand our horizons and provide a mechanism for coordination among transactions; we need solutions to support D3C! However, as we shall see, moving from an isolation-only model of transactions to D3C raises many exciting research problems touching all aspects of a database system.

3. RESEARCH CHALLENGES

In this section, we present specific technical challenges associated with supporting D3C. Our presentation is organized to match the flow of a design process for a full end-to-end solution. We start by discussing desiderata for the coordination model itself and for its efficient implementation. Next, we move on to issues that arise when integrating coordination with other database functionality such as atomic transactions and user privacy. Finally, we give some thought to architecting a database system with support for coordination.

3.1 Devising a Coordination Abstraction

Declarative Abstraction. As we have already mentioned in the introduction, the very first research challenge is to design a clean, powerful and expressive declarative model for coordination. This model should abstract away from the implementation of coordination and require users to only specify *what* coordination is required instead of specifying *how* coordination is achieved. Designing such a model will not be easy; we have already seen that there are many models for example for communication and synchronization between processes developed in other fields, and although we believe that they are not directly applicable to data-driven coordination, they provide many important design suggestions for a formal D3C model.

Meeting this challenge requires an in-depth understanding of the range of data-driven coordination tasks that users may want to perform. The model must be expressive enough to be useful in a wide range of settings; for example, it must be able to express coordination constraints beyond the simple "I want to sit next to my friend on the plane" variety. At the minimum, the abstraction must be able to express global constraints such as "No two meetings may be booked in the same time slot", or "Seats on a plane that is not full must be allocated in a way to keep it weight-balanced". Further, in auctions, donations, or other financial settings, numerical constraints such as the following are needed: "I will match all other club members' donations up to x dollars".

Programming Model. To be successful and useful, the coordination model needs more than just expressiveness. It

must be associated with a natural and intuitive programming paradigm, so that users can easily specify their coordination parameters. This has long been a challenge in the general field of concurrent programming, and it remains so for data-driven coordination. A declarative model will help in achieving clarity, as it moves complexity away from the programmer and into the system. This calls for a clean design of the actual high-level language constructs that programmers will interact with.

Queries on the Coordination State. Users may want to query the system about past and pending coordination events. For example, in a gaming scenario, a user may want to know if there are others who have expressed an intention to attack and formulate her own strategy based on this information. A user may also wish to pose such queries to understand why a coordination attempt has failed. We need to develop a suitable language for these coordination state queries that is easy to use and compatible with the coordination abstraction itself. Maybe it is even possible to represent past and pending coordinations in the system catalog and they thus become available through the standard SQL interface to the database system.

3.2 Achieving Efficient Coordination

A powerful abstraction and a declarative language for coordination are important, but neither of them will be useful without an efficient and scalable implementation. With the coordination model in place, the design of efficient algorithms to carry out the actual coordination in the system is the next challenge.

Expressiveness Versus Complexity. In the previous section, we stressed the need for the coordination model to be expressive. It is a given that with great (expressive) power comes great computational complexity. Work in some specific domains such as donation matching [2] shows clearly that the coordination problem becomes NP-complete rapidly as the expressiveness of the language increases. Therefore the development of the coordination abstraction should needs to consider practically useful tradeoff points between expressiveness and computational complexity.

Heuristics and Restrictions. In the presence of intractability, restricting expressiveness may not always be an acceptable solution. Therefore, addressing the efficiency challenge must include other strategies such as realistic restrictions on the problem instances. Such restrictions may, for example, have to do with properties of the specific workload (e.g., bounded treewidth of an underlying graph).

Efficient Algorithms. In addition to developing coordination algorithms of low computational complexity, we must search for ways to optimize them so that they scale to the large coordination settings that we imagine. There is wide scope to develop optimizations for coordination, particularly for settings with a very large number of users and coordination requests. For example, in many data-driven coordination scenarios, although the number of coordination requests in the system may be high, the requests are likely to be broadly similar (e.g., they may share query templates but differ in parameters). This will be the case, for example, with travel planning, course selection, and scheduling. Based on this observation, it may be worth exploring to what extent the coordination requests can be processed in batches rather than individually, perhaps using techniques similar to those designed for multiquery optimization [10, 3].

Another idea how to improve performance is through indexing and caching. In many scenarios (for example, in travel planning) users will not submit their coordination requests to the system simultaneously; we need a way to buffer the requests as they arrive and wait for a coordination partner. A suitable data structure can make it possible to quickly determine for an arriving coordination request whether a matching partner request exists in the system.

The feasibility of some of these optimizations will depend on where in the system coordination is implemented – i.e. whether it sits at the application layer or in the DBMS. We return to this issue in Section 3.5.

3.3 Integrating Coordination into Transactions

Concurrency control. Understanding and implementing coordination as a basic primitive is a fundamental step, but it is, however, only the first step. Generally, users expect to use coordination within larger transactions with atomicity and durability guarantees. Also, although coordination is by definition a breach of isolation, it is clear that whenever a transaction contains *non-coordinating* code, this code should be executed in isolation as far as possible. D3C must therefore be based on a concurrency control model with the power to specify exactly what to coordinate and what to isolate — and all of this without much complexity to the programmer.

Addressing this challenge involves a fundamental reassessment of the classical notion of isolation. Traditionally, isolation for transaction schedules has been defined in terms of serializability, that is, equivalence to a serial schedule. With coordinating transactions, serial execution of individual transactions is normally not possible: no single transaction may be able to proceed past a coordination step unless a partner is available.

Beyond issues of isolation, the concurrency control model needs to address issues such as the handling of dependencies which are created between transactions when they coordinate. For example, if two friends book tickets together and one of the transactions aborts, the other transaction may be required to abort as well for correctness.

Finally, once a suitable model for concurrency control is in place, it is a further challenge to figure out how to enforce it in practice via efficient protocols.

3.4 Balancing Coordination and Privacy

Coordination as an Opt-In Mechanism. While coordination inherently allows bidirectional information flow between transactions, care must be taken in designing the co-

ordination abstraction to ensure that coordination is an "opt-in" process. In many settings that we have discussed, users may want to coordinate selectively with other users and also may only want to coordinate particular aspects of their transactions. It is important that the abstraction supports such access control and prevents any unintentional information flow while still remaining flexible enough to promote coordination.

Privacy in Coordination State Queries. We have mentioned that allowing users to query the coordination state is potentially a very useful feature. However, such queries present a clear potential for privacy breaches as well; not everyone using the system may want to have their coordination requests visible to queries. Again, there is a need for a design that balances information flow and privacy; a first approach may be a careful consideration of access contol.

3.5 Architecting a System for D3C

The basic models, algorithms and protocols discussed so far are the conceptual foundation of any system that is able to support D3C. On top of this foundation, we need to build to a robust, scalable and – above all – *useful* system. This task requires a deep understanding of the architectural tradeoffs involved.

Deciding Where to Implement D3C. The most obvious tradeoff is whether the coordination should be performed within the database system itself, or outside at the application layer. Both choices come with significant advantages and disadvantages. Implementing coordination at the application layer may be simpler and easier to add to existing systems. On the other hand, as we have already discussed, the data-intensive coordination algorithms are poised to benefit significantly from deep optimizations which will be most successful if implemented at the database level.

Implications of D3C on the Architecture of a Database System. D3C as a paradigm demands a rethinking of many database fundamentals. Up until now, the need to maintain isolation among processes working on the same data has been a basic "given". As such, it permeates all aspects of the design of today's database systems for example the buffer manager, the lock manager, and query processing. When isolation is no longer guaranteed, we may benefit from re-thinking how to architect such a system. Understanding the impact on the design of a DBMS by moving from an isolation-only concurrency model for data processing to a model where both isolation and coordination are present is probably the most far-reaching research challenge of all.

4. D3C IN ACTION

In an attempt to get a handle on the challenges introduced in the previous section, we at Cornell are developing Youtopia, a prototype system that supports data-driven coordination. Here, we describe the proposed architecture of the system and a proof-of-concept application based on Youtopia that allows a user to coordinate travel plans with Facebook friends.

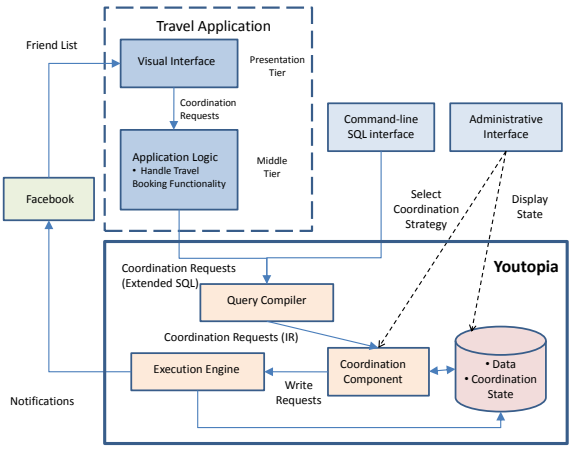


Figure 1: Draft Youtopia Architecture

Figure 2: Coordinating a Flight Booking With Friends

Figure 1 shows the proposed architecture of our system. The coordination is handled within the DBMS. Users specify which friends they want to coordinate with using a visual interface shown in Figure 2. Based on user input, formal coordination requests are generated by the middle tier and submitted to the DBMS. A query compiler processes the requests and translates them to an intermediate representation inside the DBMS for processing by the coordination component.

The coordination component runs whenever a relevant request arrives in the system, and it is able to run two different coordination algorithms. The choice of the best algorithm depends on the specific system workload; in our system this choice can be made through the administrative interface. The execution engine uses the output of the coordination algorithm and actually carries out the flight bookings to match the users' requests.

Youtopia will also provide an SQL command line interface which allows SQL and coordination requests to be input directly to the system by the user. There will also be an administrative interface which can be used to view the current state of the database tables and the coordination-related internal state information.

While our prototype is still in its early stages, our initial results are very encouraging. We have implemented the travel application on top of Youtopia, following a standard three-tier architecture. The graphical frontend runs in a browser; it is an interface to all the functionality provided by the middle tier. The middle tier contains application logic to handle the standard functionality of a travel Web site: searching for flights and hotels, selecting specific flights and hotels. The middle tier also contains code to create coordination requests based on the user's friend list, and it has access to a special Youtopia API that allows it to provide an "account view", where users can see pending or confirmed reservations.

5. CONCLUSIONS

We believe that not only people, but also database systems can be highly effective, and that it is time for them to mature beyond independence to interdependence. Data-driven coordination brings not only valuable functionality but has also set the table for a feast of challenging research problems. Let us coordinate to address these challenges!

Acknowledgments. This research was supported by the New York State Foundation for Science, Technology, and Innovation under Agreement C050061 and by the National Science Foundation under Grants 0627680 and 0725260. Any opinions, findings, conclusions, or recommendations expressed are those of the authors and do not necessarily reflect the views of the sponsors.

6. REFERENCES

[1] zope.stackless.com.

[2] V. Conitzer and T. Sandholm. Expressive negotiation over donations to charities. In *ACM Conference on Electronic Commerce*, pages 51–60, 2004.

[3] M. Hong, A. J. Demers, J. Gehrke, C. Koch, M. Riedewald, and W. M. White. Massively multi-query join processing in publish/subscribe systems. In *SIGMOD*, pages 761–772, 2007.

[4] S. L. P. Jones, A. Gordon, and S. Finne. Concurrent haskell. In *POPL*, pages 295–308, 1996.

[5] G. Koutrika, B. Bercovitz, R. Ikeda, F. Kaliszan, H. Liou, Z. M. Zadeh, and H. Garcia-Molina. Social systems: Can we do more than just poke friends? In *CIDR*, 2009.

[6] J. R. Larus and R. Rajwar. *Transactional Memory*. Morgan and Claypool, 2007.

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

[8] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, 1999.

[9] J. Reppy. *Concurrent Programming in ML*. Cambridge University Press, 1999.

[10] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. Efficient and extensible algorithms for multi query optimization. *SIGMOD Rec.*, 29(2):249–260, 2000.

[11] R. Virding, C. Wikström, and M. Williams. *Concurrent programming in ERLANG (2nd ed.)*. Prentice Hall International (UK) Ltd., Hertfordshire, UK, UK, 1996.

[12] G. Weiss. *Multiagent systems: a modern approach to distributed artificial intelligence*. MIT Press, 1999.

[13] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, 1996.

Understanding Deep Web Search Interfaces: A Survey

Ritu Khare Yuan An Il-Yeol Song
The iSchool at Drexel, Drexel University, Philadelphia, USA
{ritu, yan, isong}@ischool.drexel.edu

ABSTRACT

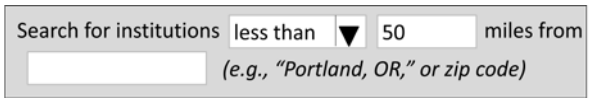
This paper presents a survey on the major approaches to *search interface understanding*. The Deep Web consists of data that exist on the Web but are inaccessible via text search engines. The traditional way to access these data, i.e., by manually filling-up HTML forms on search interfaces, is not scalable given the growing size of Deep Web. Automatic access to these data requires an automatic understanding of search interfaces. While it is easy for a human to perceive an interface, machine processing of an interface is challenging. During the last decade, several works addressed the automatic interface understanding problem while employing a variety of understanding strategies. This paper presents a survey conducted on the key works. This is the first survey in the field of search interface understanding. Through an exhaustive analysis, we organize the works on a 2-D graph based on the underlying database information extracted and based on the technique employed.

1. INTERFACE UNDERSTANDING

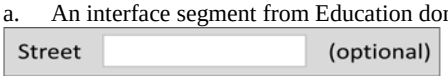
The Deep Web consists of data that exist on the Web but are inaccessible by text search engines through traditional crawling and indexing [17]. The most popular way to access these data is to manually fill-up HTML forms on search interfaces. This approach is not scalable given the overwhelming size of Deep Web [3]. Automatic access to these data has gathered much attention lately. This requires automatic understanding of search interfaces. This paper presents a survey on the major approaches to *search interface understanding* (SIU) and is the first work to present an assessment of this field.

The Deep Web is characterized by its growing scale, domain diversity, and numerous structured databases [9]. It is growing at such a fast pace that effectively estimating its size is a difficult problem [9, 19, 27, 18]. In the past, researchers have proposed many solutions to make the Deep Web data more useful to users. Ru and Horowitz [25] classify these solutions into 2 classes: dynamic content repository, and real-time search applications. We extend this classification scheme by defining 3 goal-based classes: (i) solutions to increase the content visibility on text search engines, such as collecting/indexing dynamic pages [21, 16] and creating repository of dynamic page contents [24, 29, 26]; (ii) solutions to increase the intra-domain searchability, such as meta-search engines [32, 8, 23, 5, 30, 11]; (iii) solutions to accomplish knowledge organization, such as ontology derivation [2].

Automatic understanding of Deep Web search interfaces is the pre-requisite to attain any of the aforementioned solutions. Deep Web contains at least 10 million high quality interfaces [20] and automatic retrieval of such interfaces has also received special attention [1]. Search interface understanding is the process of extracting semantic information from an interface. A search interface contains a sequence of interface components, i.e., text-labels and form elements (textbox, selection list, etc.). An interface is primarily designed for human understanding and querying. It does not have a standard layout of components (see Figure 1) and there is infinite number of possible layout patterns [7]. Thus, while human users easily perceive an interface based on past experiences and visual cues, machine processing of an interface is challenging.



a. An interface segment from Education domain



b. An interface segment from Healthcare domain

Figure 1. Diversity in Interface Design

A search interface represents a subset of queries that could be performed on the underlying Deep Web database. In data-driven Web applications, a user-specified query is translated to a structured format, such as SQL, and is executed against the online database. The placement patterns of components provide information on implied queries. For instance, the text-label 'Street:' and the adjoining textbox in Figure 1b imply the WHERE clause of an SQL query, i.e., "WHERE Street='XYZ'." Additionally, the textual contents on an interface provide information on the underlying database schema [2]. For instance, the data entering instructions such as '(eg. …' and '(optional)' in Figure 1, provide insights on data and integrity constraints, respectively.

Motivated by the opportunities provided by search interfaces, several researchers address the SIU problem. This paper presents the results of a survey conducted on the key SIU approaches. While different approaches employ different understanding strategies,

a sequence of activities is common to all. Figure 2 shows the four stages of the SIU process.

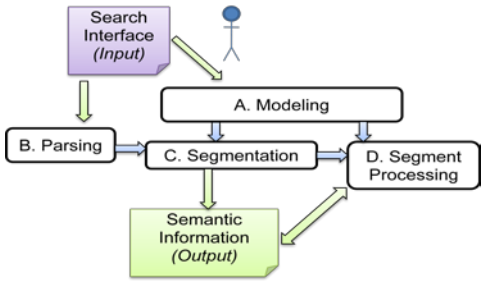


Figure 2. Search Interface Understanding Process

Input: A Search Interface
Output: Semantic Information
Stages:
(A) Modeling: First, human modelers formalize a model for the interface. Most of the works model an interface as a sequence of implied queries against the underlying database. Each query is a group of logically related components, hereafter known as a *segment.* Different approaches assign different *segment labels* to segments. For instance, Zhang et al. [33] use "conditional pattern" as the segment label and would represent Figure 3's interface as a sequence of two conditional patterns. Components in a segment might also be assigned query roles, known as *semantic labels.* Zhang et al. [33] use 3 semantic labels: "attribute-name," "operator," and "value." Furthermore, an interface might be modeled to contain constraints information about the underlying database schema. The work in [26] models information such as domain, invisible and visible values of form elements.
(B) Parsing: Then, an interface is parsed into a workable physical structure. He et al. [10] parse an interface into a string, "interface expression," with 3 constructs: 't', corresponding to any text; 'e', corresponding to any form element; and '|', corresponding to a row-delimiter. The interface in Figure 1a would be parsed as "teet|et."
(C) Segmentation: The query information, modeled in the modeling stage, is extracted in this stage. The interface is divided into segments where each segment corresponds to a query implied by the underlying database. Zhang et al. [33] use a rule-based method to group components into segments and assign semantic labels to components. The lower segment in Figure 3 is created by grouping 3 components: "Gene Name," radio button group, and textbox.
(D) Segment-Processing: This stage focuses on extracting data and integrity constraints of the underlying database. For each component, He et al. [10] extract meta-information, such as domain type and unit, using machine learning classifiers.

The semantic information identified in stages C and D is the output of the SIU process. The next section describes the settings adopted for conducting the survey.

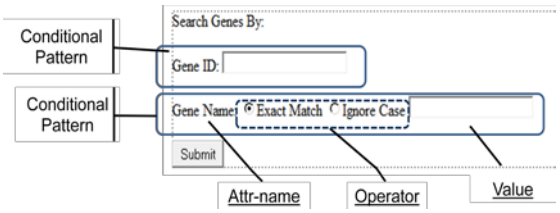


Figure 3. Segmentation by Zhang et al. [33]

2. SURVEY SETTINGS

We surveyed 10 major SIU approaches based on certain manually derived *dimensions*. A dimension is a feature of the SIU process that helps in understanding a work, and in distinguishing it with its counterparts. The survey was conducted in two phases: reductionist analysis, and holistic analysis. In the reductionist analysis phase, each work was decomposed in two ways, in terms of the four SIU stages, and in terms of the *stage-specific dimensions*. In the holistic analysis phase, each work was studied in its entirety using *composite dimensions*, created using the stage-specific dimensions. Section 3 presents the results of reductionist analysis. Section 4 presents the results of holistic analysis.

3. REDUCTIONIST ANALYSIS

We initiated the survey with the reductionist analysis phase. Each work was studied in terms of the stages of the SIU process; and in terms of the stage-specific dimensions. Each of the Sections 3.1 through 3.4 focuses on a specific stage of the process. Section 3.5 discusses the evaluation schemes employed by the surveyed approaches.

3.1 Modeling

In this stage, an interface is modeled into a formal structure suitable for machine processing. This stage was studied under the following two dimensions.

Information on Implied Queries: This dimension denotes the information related to queries implied by an interface. An interface contains multiple segments, each corresponding to an implied query. The surveyed works use a variety of segment labels to refer to a segment. The segment label adopted by *LITE* [24] and *LEX* [10] is "logical attribute." These works model an interface as a list of queries, each specific to an underlying database table attribute. Segment contents for *LITE* include a form element and a text-label. *LEX* models a segment to have a text-label, multiple form elements, and an optional text-label associated with

each form element. It assigns the semantic labels "attribute-label," "domain/constraint element," and "element label," respectively, to these components.

The work on *Hidden Syntax Parser (HSP)* in [33] adopts "conditional pattern" as the segment label. Each conditional pattern represents a query capability of the underlying database. A conditional pattern consists of a text with a semantic label "attribute name," a form element with label "operator," and a form element with label "value." The model adopted in [13] is also similar in that it represents an interface as a sequence of segments and uses "attribute-name," "operator," and "operand," as the semantic labels. Along with modeling segments corresponding to a query, Benslimane et al. [2] creates groups of segments, "structural units," each corresponding to a logical entity in the database schema.

Certain works do not explicitly assign any labels to segment or segment components, but do mention the segment contents. Kaljuvee et al. [12], *LabelEx* [22], and *DEQUE* [26] model a segment to consist of a text-label and one or more form elements. Dragut et al. [6] and *ExQ*[31] present a novel way of modeling an interface as a tree structure having arbitrary number of levels. Both these works create groups and sub-groups of related form elements and text-labels and hypothesize a hierarchical structure. Each internal node of the tree represents a text-label and has a group of related form elements as its descendants.

Hereafter, the works by Kaljuvee et al. [12], Benslimane et al. [2], Khare and An [13], and Dragut et al. [6], are referred to as *CombMatch*, *FormModel*, *HMM*, and *SchemaTree*, respectively.

Information on Constraints: This dimension denotes the information related to data and integrity constraints of the underlying database. *HSP* and *LabelEx* model a form element to have a domain of values. *DEQUE* models a form element to have domain, invisible and visible values. *LEX* models a segment to have a domain type and a default value. It models a form element to have domain type (finite, infinite, Boolean), and a unit ($, grams, days, seconds). *FormModel* includes the relationship among structural units, constraints, and the underlying source information. *HMM* models miscellaneous texts which might include information on constraints.

3.2 Parsing

Parsing marks the beginning of automatic processing and brings the interface into workable physical structure. While representation provides a logical image to an interface, parsing physically reads the interface components. Parsing strategies were studied under the following 3 dimensions.

Input Mode: The input to the parsing stage can be in two modes: HTML source code of an interface, and its visual counterpart, i.e., an interface as viewed on a Web browser. *CombMatch*, *LEX*, *FormModel*, and *HMM* use HTML code as the primary input. Along with HTML code, *LabelEx*, *LITE*, *HSP*, *DEQUE*, *ExQ*, and *SchemaTree* use layout engines to extract the visual features such as pixel distances between components.

Description: This dimension refers to the tasks performed while parsing an interface. *LITE* parses an interface in the "Pruning" stage wherein the components that directly affect the layout and labels of form elements are isolated from the rest. *CombMatch*, in its "Chunk Partitioning" stage, segments an interface into chunks delimited by HTML and TABLE cell tags. *LEX* develops an "interface expression" that looks like 't|eee|te|ee|ttee|eet|'. *HSP* parses a page into a set of tokens using its module, "Tokenizer," and stores information such as name, layout position, etc. *HMM* creates a DOM tree of interface components and traverses the tree in depth-first order. *SchemaTree*, in its "Token Extraction" module, creates lists of text tokens, field tokens, and image tokens, and also stores the information about their bounding boxes.

Purgation: This dimension enlists the components that are removed while parsing to avoid information overload on subsequent stages. *LITE* discards images and text styling information. *FormModel* and *CombMatch* remove stop words and text formatting tags. *DEQUE* ignores the components that correspond to font size, typefaces and styling information. *HMM* ignores all the components except the form elements and the text-labels.

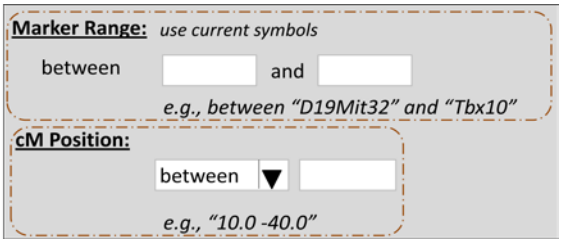


Figure 4. Segmented Search Interface

3.3 Segmentation

After a suitable logical representation and a physical structure are accomplished, the interface is segmented, i.e., the information regarding the implied queries is extracted from the interface. Figure 4 shows a segmented interface having 2 queries. This stage was studied under the following dimensions.

Segmentation Tasks: Segmentation can be visualized as a 3-task process. The first task, *text-label assignment,* involves associating a form element with a surrounding text-label, e.g., associating "cM Position:" with the form elements, selection list, and textbox, in the bottom segment of Figure 4. The second task is *grouping* where the related interface components are grouped together to form a segment. In Figure 4, the 4 components ('cM Position:,' selection list, textbox, and 'e.g., "10.0-40.0"') belong to the same atomic query and are hence grouped together. In the third task, *semantic labeling,* labels or query roles are assigned to individual components of a query. Automatic text-label assignment and grouping are difficult due to diversity in Web design. Automatic semantic labeling is difficult as Web designers usually do not assign explicit labels in the HTML source code.

A majority of the works (*LITE, CombMatch, DEQUE, LabelEx*) only address the text-label assignment problem. *LEX* groups related text-labels and form elements together into "logical attributes". *HSP* finds groups of "conditional patterns." *LEX, HSP,* and *HMM,* perform grouping as well as semantic labeling. *LEX* also identifies the "exclusive attributes" on an interface based on a domain-specific vocabulary. *SchemaTree* performs text-label assignment and creates segments and sub-segments resulting into a tree of interface tokens. *ExQ* extracts the grouping information of an interface into an unlabeled tree structure and then performs text-label assignment to generate a labeled tree.

Segmentation Technique: Segmentation techniques, i.e., the mechanisms to segment an interface, belong to 3 categories: heuristics, rules, and machine learning.

Heuristic-properties are of 3 kinds: textual, styling and layout. Textual properties include text length, no. of words, string similarity, element's HTML name, etc. Styling properties include font size, font type, form element format, etc. Layout properties include position of a component, distance between two components, etc. To perform text-label assignment *LITE* exploits all 3 kinds of heuristics. *CombMatch* uses a combination of 8 different algorithms leveraging the 3 kinds of heuristics to assign text-label to a form element. *DEQUE* and *LEX* perform text-label assignment based on the textual and layout properties of components. In *LEX,* all the form elements associated with same text and the text itself are assigned to one segment. Based on heuristics, it also assigns the semantic labels, "attribute label," "constraint element," "domain element," and "element label" to the components.

A rule is a formalized heuristic. Rule-based techniques employ techniques such as regular expressions, grammar, finite state methods, and create rules for associating a form element with a surrounding text. *HSP* assumes that a hidden syntax guides the presentation of interface components on a query interface. The identification of segments and semantic labels is performed using a grammar. The grammar rules are based on layout properties and are derived using pre-studied examples.

SchemaTree uses both rules and heuristics. A tree of fields is built based on the layout properties of form elements, and a tree of text tokens is built based on the layout and styling properties of the text-labels. Then, the two trees are integrated based on some common-sense rules, to generate a complete schema tree corresponding to the interface.

Recent years have seen an advent of machine learning techniques in the field of interface understanding. *LabelEx* employs supervised machine learning to assign labels to form elements. It designs a "Classifier Ensemble" using Naïve Bayes and Decision Trees classifiers and employs both textual and layout properties to perform text-label assignment. *HMM* explores another machine learning technique, Hidden Markov Models. It creates a 2-layered artificial designer having the ability to understand an interface based on the layout and textual properties of components. The first layer tags the components with semantic labels, and the second layer identifies the boundaries of segments. *ExQ* creates the interface structure tree using hierarchical agglomerative spatial clustering. Each form element is considered to be a visual attribute block. To generate the tree, spatially closer and similarly styled blocks are clustered under the same internal node. *ExQ* performs node label assignment using annotation rules and hence falls under a hybrid category.

3.4 Segment Processing

After an interface is segmented, more semantics related to segments and segment components are extracted. This includes information on data and integrity constraints of the underlying database. While several approaches enlist this information in the modeling stage, very few extract it. These approaches were studied under the following dimensions.

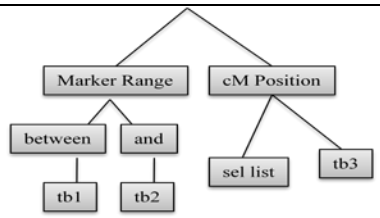
Technique: *LEX* uses machine learning classifiers to identify more semantics from a segment, such as type, domain type, value type, unit of form elements, relationship and semantics of domain elements, and logic relationship of attributes. *FormModel* uses another machine learning technique, learning by

example, to extract relationship between two "structural units," and constraints of a form instance.

Post-processing: *LITE* and *LEX* post-process the text-labels by removing stop words such as "the," "any," etc. *LITE* also performs standard IR-style stemming on the text-labels. *HSP*'s "Merger" module reports conflicting tokens that occur in more than one query conditions, and missing tokens that they do not occur in any query condition. *LabelEx* devises heuristics for reconciliation of multiple labels assigned to an element and for handling form elements with unassigned labels.

Table 1 gives a summarized view of reductionist analysis showing the outputs generated by each work as a result of understanding the interface in Figure 4.

Table 1. Summary of Reductionist Analysis

	Modeling	*Parsing*	*Segmentation & Processing*	*Semantic Information*
CombMatch, LITE LabelEx DEQUE	Logical Attribute = <text-label, form element(s)>	Chunk Partitioning (CombMatch), Pruning (LITE).	tb1 => between, tb2 => and, select list => cM Position: , tb3 => cM Position:	4 label assignments
HSP	Pattern= <attr-name, operator, value>	Token positions.	<between, tb1>, <and, tb2>, <cM Position, sel list, tb3>	3 query conditions
LEX	Logical Attribute = <attr-label, element label, domain/ constraint element, domain type, default value>	Interface expression: tt|tete|t|t|ee|t	{Attr-label = Marker Range, Ele-label = between, Domain element = tb1, Ele-label = and, Domain element = tb2} {Attr-label = cM Position, Const. element = selection list, Domain element = tb3}	2 logical attributes
HMM	Segment= <attr-name (s), operator(s), operantor(s), misc-text(s)>	Pre-order DOM traversal: Marker…, use…, between, tb1, and, tb2, e.g. between, cM Position, sel list, tb3, e.g "10.0-40.0"	{Attr-name = Marker Range, operator = between, operand = tb1, operator = and, operand = tb2, Misc-texts = use current …, e.g., bet …} {Attr-name = cM Position, operator = selection list, Misc-texts = e.g., "10.0 – 40.0"}	2 segments
SchemaTree ExQ	Tree Node = text-label or form element.	Text tokens: Marker Range:, use current, between, … Field tokens: tb1, tb2,… & bound. boxes for all tokens) (*SchemaTree* only)		1 tree

3.5 Evaluation

Although evaluation is not a part of the core SIU process, it acts as a significant after-stage in all surveyed approaches. Here, the extracted semantic information is evaluated by comparing with either the manually extracted information or a gold standard as in the cases of *SchemaTree, LabelEx,* and *ExQ*.

Test Domain: The surveyed approaches are tested on several domains. The most popular choices of researchers are automobile, airfare, books, movies and real estate, followed by car rental, hotel, music, and jobs. Some of the least tested domains include biology, database technology, electronics, games, health, medical, references and education, scientific publication, semiconductors, shopping, toys, and watches. We compiled a list of various datasets at http://cluster.ischool.drexel.edu:8080/ibiosearch/datasets html.

Metrics: *LITE, HMM,* and *LEX* report the extraction accuracy, i.e., the number of correctly identified components (segments) over the total number of manually identified components (segments). *DEQUE* reports the label extraction accuracy and the domain value extraction accuracy. *CombMatch* reports the success percentage, i.e., the number of correctly identified text-labels over the total number of elements, and the failure percentage, i.e., the number of incorrectly identified text-labels over the total number of elements. *HSP* reports precision and recall. Precision is the number of correctly identified segments over the total number of identified segments. Recall is the number of correctly identified segments over the total number of manually identified segments. *LabelEx* also reports recall, precision, and F-measure. *SchemaTree* measures text-label assignment accuracy, and the overall precision, recall and F-score. *ExQ*

measures precision and recall for grouping, ordering, and node labeling.

Comparison of performance: Most of the surveyed works evaluate the performance by comparing their results with those of one or more of the contemporary works. *HSP* and *LEX* are the most widely used benchmarks for evaluation of performance. *HSP* was chosen by *LEX*, *LabelEx*, and *SchemaTree*, to compare the performances of respective works; and *LEX* was chosen by *LabelEx*, *SchemaTree*, and *HMM*. Another benchmark work is *CombMatch*, chosen by *LITE*.

4. HOLISTIC ANALYSIS

Section 3 viewed each work in the light of the stage-specific dimensions. It was found that certain dimensions, such as query and constraint information, segmentation task, and segmentation and segment processing technique, hold more potential for making significant changes in the overall process. These dimensions were used to create two composite dimensions for holistic analysis: database description and extraction technique. Based on this, the surveyed approaches can be plotted on a 2-D graph (See Figure 5) with the two axes corresponding to the two composite dimensions.

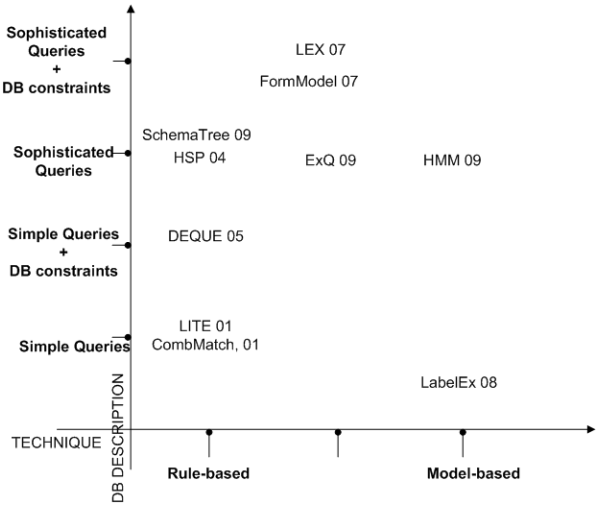
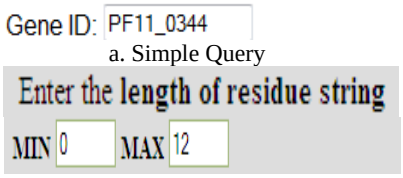


Figure 5. Holistic Analysis

Database Description: This dimension is described along the Y-axis and denotes the underlying database information extracted by a given approach. The surveyed approaches can be organized into 4 levels. The first level consists of *LITE*, *CombMatch*, and *LabelEx*. These works extract simple queries by performing text-label assignment. Figure 6a shows an example of a simple query extractible by associating "Gene ID:" with the adjoining textbox. This corresponds to the clause, "WHERE GeneID = 'PF11_0344.'" However, text-label assignment at times results in extraction of partial query capabilities when it faces sophisticated designs like the one shown in Figure 6b. Such works might assign both textboxes to the text-label "Enter the length …," but would fail to extract the complete implied query that corresponds to the clause, "WHERE length>=0 AND length <=12." At the next level lies the work *DEQUE*. This approach extracts simple query capabilities along with data and integrity constraints of the underlying database.

The next level includes the works that extract sophisticated queries, like the one in Figure 6b, from an interface. *HSP*, *LEX*, and *HMM* identify such queries by grouping all related components into segments corresponding to logical attributes. *FormModel* forms a different type of segment that refers to an entity, "structural unit," instead of an attribute. *SchemaTree* and *ExQ* are different too in that they perform hierarchical grouping and the queries extracted might be associated with both attributes and entities. Both *LEX* and *FormModel* employ strategies for extracting data and integrity constraints too, and thus, occur at the highest level.

Extraction Technique: This dimension refers to the techniques employed during the stages, segmentation and segment processing. These techniques fall under two categories: rules and models. We blend rules and heuristics into the rule-based category, and supervised and unsupervised machine learning into the model-based category. *HSP*, *LITE*, *CombMatch*, *DEQUE* and *SchemaTree* represent the rule-based approaches. *LabelEx* and *HMM* are both model-based. *LEX* and *FormModel* lie in between the two categories because they extract implied queries using rules, and extract constraint information using models. *ExQ* too lies in between as it performs grouping using a clustering model and performs text-label assignment using rules.



a. Simple Query

b. Sophisticated Query

Figure 6. Types of Queries

Holistic analysis reveals two striking points regarding the journey of interface understanding in the past decade. First, a considerable progress has been made in terms of the underlying database information extracted. This is depicted by the transition from simple to sophisticated query capabilities across the Y-axis of the graph. However, the extracted information on data and integrity constraints does not appear to follow a regular timeline. Secondly, an improvement in the

sophistication level of segmentation and segment-processing techniques, from rule-based to model-based techniques, is clearly visible along the X-axis.

5. OPEN QUESTIONS

The survey on the key search interface understanding approaches helped in identifying several unaddressed issues in this field. In terms of database description, we are very far from extracting the complete schema of the database that lies underneath an interface. In terms of the employed technique, previous studies [14, 15] have favored model-based over rule-based approaches for handling design heterogeneity. This is followed by a logical transition from rules to models. However, this transition did not have much effect on the degree of human intervention. While rule-based approaches, such as *HSP* and *LITE*, require manual specification of rules and human observations of heuristics, the model-based approaches, such as *LabelEx* and *HMM*, require manual annotation of the training data. *ExQ* made the first step toward unmediated understanding by employing a clustering technique to derive the initial tree structure for an interface. Such unsupervised learning techniques are much needed for developing scalable SIU solutions.

It should be noted that this survey focused on those approaches that attempt to understand an interface solely based on the information available on the interface itself. Interestingly, there is another alternative of deriving interface semantics, which is, by filling up the HTML forms using instances and analyzing the result pages. For example, [30] performs text-label assignment by "query probing," and [24] derives the domain of form element values using form submissions. Also, in the quest of surfacing, the work in [21] determines whether a form element is a "binding" or a "free" input, by generating the result pages. Another work [28] determines a list of possible "atomic queries" for an interface using form submissions. An "atomic query" is a minimal set of attributes that result in a valid result page. A combination of both interface-based and instance-based approaches of form understanding has not yet been explored. It should also be noted that certain works [23, 32] that perform SIU were not included in this survey. These works manually extract semantic information from interfaces and thus could not contribute much to the discussion of holistic and reductionist analysis.

Based on our findings in Section 3.5, a majority of the tested domains fall under the commercial Yahoo subject categories [9]. The other half of the Deep Web, containing databases from non-commercial domains [9] such as education, arts, science, reference, etc., has hardly been explored. Previous studies [13, 22, 33] have investigated the question of whether an SIU approach should be domain-specific or generic. Considering that a significant number of domains have remained unexplored and that the interface designs differ across subject domains, this question needs to be re-investigated on a balanced dataset of commercial and non-commercial domains.

Lastly, most of the SIU approaches have been designed for a specific application. While *HSP*, *LEX*, *LabelEx*, and *SchemaTree* target to increase the intra-domain usability of Deep Web contents, *DEQUE*, *LITE*, *LabelEx*, and *ExQ* target to increase content visibility on text search engines. Out of all, *SchemaTree* shines out as it has been cautiously designed to suit specific applications like interface matching and unification. This suggests the importance of aligning methodologies with intended applications. In future, a formal study of the correlation between the extraction methodologies and the potential application will be greatly beneficial.

6. REFERENCES

[1] Barbosa, L., Tandon, S., and Freire, J. 2007. Automatically constructing a directory of molecular biology databases. In Proc. of the International Workshop on Data Integration in the Life Sciences (Philadelphia, PA, Jun. 27-29, 2007) DILS' 07. Springer Berlin, Heidelberg. 6-16.

[2] Benslimane, S. M., Malki, M., Rahmouni, M. K., and Benslimane, D. 2007. Extracting personalised ontology from data-intensive web application: An HTML forms-based reverse engineering approach. Informatica, 18, 4 (Dec. 2007), 511-534.

[3] Bergman, M., K. 2001. The deep web: Surfacing hidden value. White Paper. University of Michigan.

[4] Cawsey, A. 1998. The essence of artificial intelligence Prentice Hall, Upper Saddle River, NJ.

[5] Chang, K. C., He, B., and Zhang, Z. 2005. Toward large scale integration: Building a MetaQuerier over databases on the web. In Proc. of the 2nd Conference on Innovative Data Systems Research (Asilomar, CA, Jan. 4-7, 2005) CIDR'05. ACM Press, New York, NY. 44-55.

[6] Dragut, E., C., Kabisch, T., Yu, C., and Leser, U., 2009. A Hierarchical Approach to Model Web Query Interfaces for Web Source Integration. In Proc. of the 35th International Conference on Very Large Data Bases (Lyon, France, August 24-28, 2009). VLDB'09. IEEE Computing Society, Washington, DC, 325 - 335.

[7] Halevy, A. Y. 2005. Why your data won't mix: Semantic heterogeneity. Queue, 3, 8(Oct. 2005), 50-58.

[8] He, B., and Chang, K. C. 2003. Statistical schema matching across web query interfaces. In Proc. of the ACM International Conference on Management of Data (San Diego, CA, June 9-12, 2003). SIGMOD'03. ACM Press, New York, NY. 217-228.

[9] He, B., Patel, M., Zhang, Z., and Chang, K. C. 2007. Accessing the deep web. Communications of the ACM, 50, 5 (Oct. 2008), 94-101.

[10] He, H., Meng, W., Lu, Y., Yu, C., and Wu, Z. 2007. Towards deeper understanding of the search interfaces of the deep web. World Wide Web, 10,2 (Jun. 2007), 133 - 155.

[11] He, H., Meng, W., Yu, C., and Wu, Z. 2004. Automatic integration of web search interfaces with WISE-integrator. The VLDB Journal the International Journal on very Large Data Bases, 13, 3(Sep. 2004), 256-273.

[12] Kaljuvee, O., Buyukkokten, O., Garcia-Molina, H., and Paepcke, A. 2001. Efficient web form entry on PDAs. In Proc. of the 10th International Conference on World Wide Web (Hong Kong, China, May 1-5, 2001). WWW'01. ACM Press, New York, NY, 663 - 672.

[13] Khare, R., and An, Y. 2009. An Empirical Study On Using Hidden Markov Model for Search Interface Segmentation. In Proc. of the 18th International Conference on Information and Knowledge Management (Hong Kong, China, Nov 2-6, 2009). CIKM'09. ACM Press, New York, NY, 17 -26.

[14] Kushmerick, N. 2002. Finite-state approaches to web information extraction. 3rd Summer Convention on Information Extraction (Frascati, Italy, July 15-19, 2002) SCIE'02, Springer, Berlin, Heidelberg, 77-91.

[15] Kushmerick, N. 2003. Learning to invoke web forms. In On the move to meaningful internet systems. Springer Berlin, Heidelberg, 997-1013.

[16] Lage, J. P., da Silva, A. S., Golgher, P. B., and Laender, A. H. F. 2004. Automatic generation of agents for collecting hidden web pages for data extraction. Data and Knowledge Engineering, 49, 2(May 2004), 177 - 196.

[17] Lawrence, S., and Giles, C. L. 1998. Searching the World Wide Web. Science, 280, 5360(Apr. 1998), 98-100.

[18] Ling, Y., Meng, X., and Liu, W. 2008. An attributes correlation based approach for estimating size of web databases. Journal of Software, 19, 2(Mar/Apr. 2007), 224-236.

[19] Lu, J. (2008). Efficient estimation of the size of text deep web data source. In Proc. of the 17th ACM Conference on Information and Knowledge Management (Napa Valley, CA, Oct. 26-30, 2008) CIKM '08. ACM Press, New York, NY.

[20] Madhavan, J., Jeffery, S., R., Cohen, S. I., Dong, X., Ko, D., and Yu, C. 2007. Web-scale data integration: You can only afford to pay as you go. In Proceedings of Conference on Innovative Data Systems Research (Pacific Grove, CA, Jan. 7-10, 2007) CIDR'07. ACM Press, New York, NY. 40-48.

[21] Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., and Halevy, A. Y. 2008. Google's deep web crawl. Proc. of the VLDB Endowment, 1, 2 (Aug. 2008), 1241-1252.

[22] Nguyen, H., Nguyen, T., and Freire, J. 2008. Learning to extract form labels. In Proceedings of the VLDB Endowment, Auckland, New Zealand. , 1, 1(Aug. 2008), 684-694.

[23] Pei, J., Hong, J., and Bell, D. 2006. A robust approach to schema matching over web query interfaces. In Proc. of the 22nd International Conference on Data Engineering Workshops (Atlanta, GA, April 3-7, 2006). ICDEW'06. IEEE Computer Society, Washington, DC. 46-55.

[24] Raghavan, S., and Garcia-Molina, H. 2001. Crawling the hidden web. In Proc. of the 27th International Conference on very Large Data Bases (Rome, Italy, September 11-14, 2001) VLDB '01, Morgan Kaufmann Publishers Inc, San Francisco, CA, 129-138.

[25] Ru, Y., and Horowitz, E. 2005. Indexing the invisible web: A survey. Online Information Review, 29, 3 (Apr. 2005), 249-265.

[26] Shestakov, D., Bhowmick, S., S., and Lim, E. 2005. DEQUE: Querying the deep web. Data and Knowledge Engineering, 52, 3 (Mar. 2005), 273-311.

[27] Shestakov, D., and Salakoski, T. 2007. On estimating the scale of national deep web. Database and expert systems applications, Springer Berlin, Heidelberg, 780-789.

[28] Shu, L., Meng, W., He, H., and Yu, C. 2007. Querying Capability Modeling and Construction of Deep Web Sources. In Proc. of 8th International Conference on Web Information Systems Engineering (Nancy, France, December 3-6, 2007) WISE '07. Springer Berlin, Heidelberg, 13-25.

[29] Wang, J., and Lochovsky, F. 2003. Data extraction and label assignment for web databases. In Proc. of 12th International Conference on World Wide Web (Budapest, Hungary, May 20-24, 2003) WWW '03. ACM Press, New York, NY, 187-196.

[30] Wang, J., Wen, J., Lochovsky, F., and Ma, W. 2004. Instance-based schema matching for web databases by domain-specific query probing. In Proc. of 30th International Conference on Very Large Data Bases (Toronto, Canada, August 29-30, 2004) VLDB '04, VLDB Endowment, 408 - 419.

[31] Wu, W., Doan, A., Yu, C., and Meng, W. 2009. Modeling and Extracting Deep-Web Query Interfaces. In Advances in Information and Intelligent Systems. Springer Berlin, Heidelberg, 65-90.

[32] Wu, W., Yu, C., Doan, A., and Meng, W. 2004. An interactive clustering-based approach to integrating source query interfaces on the deep web. In Proc. of the ACM International Conference on Management of Data (Paris, France, June 13-18, 2004) SIGMOD '04.ACM, New York, NY, 95 - 106.

[33] Zhang, Z., He, B., and Chang, K. C. 2004. Understanding web query interfaces: Best-effort parsing with hidden syntax. In Proc. of the ACM International Conference on Management of Data (Paris, France, June 13-18, 2004) SIGMOD '04.ACM, New York, NY, 107 – 118.

Search Result Diversification

Marina Drosou
Dept. of Computer Science
University of Ioannina, Greece
mdrosou@cs.uoi.gr

Evaggelia Pitoura
Dept. of Computer Science
University of Ioannina, Greece
pitoura@cs.uoi.gr

ABSTRACT

Result diversification has recently attracted much attention as a means of increasing user satisfaction in recommender systems and web search. Many different approaches have been proposed in the related literature for the diversification problem. In this paper, we survey, classify and comparatively study the various definitions, algorithms and metrics for result diversification.

1. INTRODUCTION

Today, most user searches are of an exploratory nature, in the sense that users are interested in retrieving pieces of information that cover many aspects of their information needs. Therefore, recently, *result diversification* has attracted considerable attention as a means of counteracting the over-specialization problem, i.e. the retrieval of too homogeneous results in recommender systems and web search, thus enhancing user satisfaction (e.g. [20, 16]). Consider, for example, a user who wants to buy a car and submits a related web search query. A diverse result, i.e. a result containing various brands and models with different horsepower and other technical characteristics is intuitively more informative than a result that contains a homogeneous result containing only cars with similar features.

Diversification is also useful in counter-weighting the effects of personalization. Personalization aims at tailoring results to meet the preferences of each specific individual (e.g. [10, 15]). However, this may lead to overly limiting the search results. Diversification can complement preferences and provide personalization systems with the means to retrieve more satisfying results (as in [14]).

In this paper, we survey the various approaches taken in the area of result diversification. We classify the ways that diverse items in the related literature are generally defined in three different categories, namely in terms of (i) *content* (or *similarity*), i.e. items that are dissimilar to each other (e.g. [18]), (ii) *novelty*, i.e. items that contain new information when compared to previously seen ones (e.g. [3, 19]) and (iii) *coverage*, i.e. items that belong to different categories (e.g. [1]). Then, we present various algorithms for result diversification and classify them into two main groups, namely (i) *greedy* (e.g. [20]) and (ii) *interchange* (e.g. [17]) algorithms. We also show the main metrics used for evaluating the performance of diversification systems.

The rest of this paper is structured as follows. In Section 2, we classify various definitions of the result diversification problem, while in Section 3, we see how diversity is combined with other ranking criteria. In Section 4, we review the proposed algorithms for efficiently retrieving diverse results and, in Section 5, we show measures used for evaluating the diversity of selected items. Finally, Section 6 concludes this paper.

2. DIVERSITY DEFINITIONS

Generally, the problem of selecting diverse items can be expressed as follows. Given a set[1] $\mathcal{X}$ of n available items and a restriction k on the number of wanted results, the goal is to select a subset S^* of k items out of the n available ones, such that, the diversity among the items of S^* is maximized.

In this section, we present various specific definitions of the result diversification problem that can be found in the research literature. We classify these definitions based on the way that diverse items are defined, i.e. (i) content, (ii) novelty and (iii) coverage. Note that, this classification is sometimes fuzzy, since these factors are related to each other and, therefore, a definition can affect more than one of them.

2.1 Content-based definitions

Content-based definitions interpret diversity as an instance of the *p-dispersion problem*. The ob-

[1]In some works, the term "set" is used loosely to denote a set with *bag semantics* or a *multiset*, where the same item may appear more than once in the set.

jective of the p-dispersion problem is to choose p out of n given points, so that the minimum distance between any pair of chosen points is maximized [6]. The p-dispersion problem has been studied in the field of Operations Research for locating facilities that should be dispersed; such as franchises belonging to a chain or nuclear power plants. Formally, the p-dispersion problem is defined as follows:

Given a set $\mathcal{X}$ of points, $\mathcal{X} = \{x_1, \ldots, x_n\}$, a distance metric $d(.,.)$ among points and an integer k, locate a subset S^* of $\mathcal{X}$, such that:

$$S^* = \underset{\substack{S \subseteq \mathcal{X} \\ |S|=k}}{\operatorname{argmax}} f(S), \text{ where } f(S) = \min_{\substack{x_i, x_j \in \mathcal{X} \\ x_i \neq x_j}} d(x_i, x_j)$$

Content-based definitions of diversity have been proposed in the context of web search and recommender systems. Most often, however, the objective function that is maximized is the *average distance* of any two points, instead of the minimum one, that is:

$$f(S) = \frac{2}{k(k-1)} \sum_{i=1}^{k} \sum_{j>i}^{k} d(x_i, x_j) \qquad (1)$$

This approach is followed in [20], where the diversity of a set of recommendations in a typical recommender system is defined based on their *intra-list similarity*, which is the application of Equation 1 along with a user-defined distance metric.

Another work that defines diverse recommendations based on content is [17]. The distance between recommendations is measured based on their *explanations*. Given a set of items $\mathcal{X}$ and a set of users $\mathcal{U}$, the explanation of an item $x \in \mathcal{X}$ recommended to a user $u \in \mathcal{U}$ can be defined in a content-based approach as:

$$Expl(u, x) = \{x' \in \mathcal{X} | sim(x, x') > 0 \wedge x' \in Items(u)\}$$

where $sim(x, x')$ is the similarity of x, x' and $Items(u)$ is the set of all items rated in the past by user u. A non content-based collaborative filtering approach is also considered, in which:

$$Expl(u, x) = \{u' \in \mathcal{U} | sim'(u, u') > 0 \wedge x \in Items(u')\}$$

where $sim'(.,.)$ is a similarity metric between two users. The similarity $sim(.,.)$ between two items x and x' can be defined based on the Jaccard similarity coefficient, the cosine similarity or any other similarity measure. The diversity of a set of items $S \subseteq \mathcal{X}$ is defined as the average distance of all pairs of items (as in Equation 1). A similar Jaccard-based similarity measure is also used in [7]. In that case, each document is described by a sketch produced by a number of hash functions. Another alternative distance metric used in that work is a taxonomy-based categorical distance when this can be applied (e.g. in the case of documents).

A content-based definition of diversity has also been applied in the context of publish/subscribe systems [5, 4]. Here, given a period or a window of matching events and an integer k, only the k most diverse of them (based on Equation 1) are delivered to the related subscribers.

Another definition that can be classified in this category is the one used in [16] in the context of database systems. Given a database relation $\mathcal{R} = (A_1, \ldots, A_m)$, a *diversity ordering* of $\mathcal{R}$, denoted $\prec_{\mathcal{R}}$, is a total ordering of its attributes based on their importance, say $A_1 \prec \ldots \prec A_m$. Also, a *prefix with respect to* $\prec_{\mathcal{R}}$, denoted ρ, is defined as a sequence of attribute values in the order given by $\prec_{\mathcal{R}}$, moving from higher to lower priority. Let ρ be a prefix of length l and t, t' be two tuples of $\mathcal{R}$ that share ρ. The similarity between t and t' is defined as:

$$sim_\rho(t, t') = \begin{cases} 1 & \text{if } t.A_{l+1} = t'.A_{l+1} \\ 0 & \text{otherwise} \end{cases}$$

Now, given an integer k, a subset S of $\mathcal{R}$ with cardinality k is defined to be diverse with respect to ρ if all tuples in S share the prefix ρ and the sum of their pair-wise similarities, as defined above, is minimized. S is also said to be diverse with respect to $\mathcal{R}$ if it is diverse with respect to every possible prefix for $\mathcal{R}$.

Finally, a content-based definition of diversity is used in [8] to extend the *k-nearest neighbor* problem, so that, given an item x, the k spatially closest results that are sufficiently different from the rest of the answers are retrieved. In this case, the distance between two items is based on the Gower coefficient, i.e. a weighted average of the respective attribute differences of the items. Assuming δ_i to be equal to the difference between the i^{th} attributes of two items x, x' then:

$$d(x, x') = \sum_i w_i \delta_i$$

where w_i is a weight corresponding to the i^{th} dimension of the items. Two items are considered diverse if their distance $d(.,.)$ is greater than a given threshold and a set S is considered diverse if all the pairs of items in it are diverse.

2.2 Novelty-based definitions

Novelty is a notion closely related to that of diversity, in the sense that items which are diverse from all items seen in the past are likely to contain novel information, i.e. information not seen before.

A distinction between novelty and diversity in the context of information retrieval systems is made in

[3], where novelty is viewed as the need to avoid redundancy, whereas diversity is viewed as the need to resolve ambiguity. Each document x and query q are considered as a collection of *information nuggets* from the space of all possible nuggets $\mathcal{O} = \{o_1, \ldots, o_m\}$. Given a binary random variable R_x that denotes whether a document x is considered relevant to a given query q, then:

$$P(R_x = 1|q, x) = P(\exists o_i, \text{ such that } o_i \in x \cap q)$$

Now, given an ordered list of documents $x_1, \ldots, x_n$ retrieved by an IR system for q, the probability that the k^{th} document is both novel and diverse from the $k-1$ first ones, i.e. $R_{x_k} = 1$, is equal to the probability of that document containing a nugget that cannot be found in the previous $k-1$ documents. Given a list of $k-1$ preceding documents, the probability that a nugget $o_i \in \mathcal{O}$ is novel for a query q is:

$$P(o_i \in q|q, x_1, \ldots, x_{k-1}) = P(o_i \in q) \prod_{j=1}^{k-1} P(o_i \notin x_j)$$

Assuming that all nuggets are independent and equally likely to be relevant for all queries, then:

$$P(R_{x_k} = 1|q, x_1, \ldots, x_k) = 1 - \prod_{i=1}^{m} (1 - \gamma\alpha J(x_k, o_i)(1-\alpha)^{r_{o_i,k-1}}) \quad (2)$$

where $J(x_k, o_i) = 1$ if some human judge has determined that x_k contains the nugget o_i (or zero otherwise), α is a constant in $(0, 1]$ reflecting the possibility of a judge error in positive assessment, $\gamma = P(o_i \in q)$ and $r_{o_i,k-1}$ is the number of documents ranked up to position $k-1$ that have been judged to contain o_i, i.e. $r_{o_i,k-1} = \sum_{j=1}^{k-1} J(x_j, o_i)$. This approach requires prior knowledge of the nuggets and also considerable amount of human effort for judging the relevance of documents in order to compute the related probabilities.

Another work based on novelty is [19], which aims at enhancing adaptive filtering systems with the capability of distinguishing novel and redundant items. Such systems should identify documents that are similar to previously delivered ones, in the sense of having the same topic, but also dissimilar to them, in the sense of containing novel information. The redundancy R of each document x is measured with respect to its *similarity* to all previously delivered documents, denoted $D(x)$, as follows:

$$R(x|D(x)) = \operatorname*{argmax}_{x' \in D(x)} R(x|x')$$

where $R(x|x')$ is the redundancy (similarity) of x with respect to another document x'. Three different ways for measuring $R(x|x')$ are considered, namely the *set difference*, the *geometric distance* and the *distributional distance*. The set difference is based on the number of new terms that appear in x:

$$R(x|x') = \left| Set(x) \cap \overline{Set(x')} \right|$$

In the above formula, given a term w and a document x, it holds that $w \in Set(x)$, if and only if, $Count(w, x) > h$, where h is a constant and $Count(w, x) = \alpha_1 tf_{w,x} + \alpha_2 df_w + \alpha_3 rdf_w$. $tf_{w,x}$ is the frequency of w in x, df_w is the number of all filtered documents that contain w, rdf_w is the number of delivered documents that contain w and $\alpha_1, \alpha_2, \alpha_3$ are constants with $\alpha_1 + \alpha_2 + \alpha_3 = 1$. The geometric distance is based on the cosine similarity between x and x': If we represent each document x as a vector $\mathbf{x} = (tf_{w_1,x}, tf_{w_2,x}, \ldots, tf_{w_m,x})^T$, where $w_1, w_2, \ldots, w_m$ are all the available terms, then:

$$\begin{aligned} R(x|x') &= \cos(\mathbf{x}, \mathbf{x}') \\ &= \frac{\mathbf{x}^T \mathbf{x}'}{\|\mathbf{x}\| \, \|\mathbf{x}'\|} \end{aligned}$$

Finally, the distributional distance is based on a probabilistic language model. Each document x is represented by a unigram word distribution θ_x and the distance among two documents is measured via the Kull- back-Leibler formula:

$$\begin{aligned} R(x|x') &= -KL(\theta_x, \theta_{x'}) \\ &= -\sum_{w_j} P(w_j|\theta_x) \log \frac{P(w_j|\theta_x)}{P(w_j|\theta_{x'})} \end{aligned}$$

A mixture-model approach is considered in order to find the language models for the θ distributions.

2.3 Coverage-based definitions

Some works view diversity in a different way, that of selecting items that cover many different interpretations of the user's information need. For example, [1] considers typical web search and, given a query q and a taxonomy C of independent information categories, aims at retrieving k documents that cover many interpretations of q, especially interpretations that are considered important. The result diversification problem in this context is formally defined as follows: Given a query q, a set of documents $\mathcal{X}$, a taxonomy C, a probability distribution $P(c|q)$ of each category $c \in C$ being relevant to q, the probability $V(x|q, c)$ of each document $x \in \mathcal{X}$ being relevant to each category c for q and an integer k, find a set of documents S^*, such that:

$$S^* = \operatorname*{argmax}_{\substack{S \subseteq \mathcal{X} \\ |S|=k}} P(S|q)$$

where:

$$P(S|q) = \sum_c P(c|q)(1 - \prod_{x \in S} (1 - V(x|q, c))) \quad (3)$$

The probability of x *not* covering a relevant to the query q category c is equal to $(1-V(x|q,c))$. Therefore, the above equation, in essence, maximizes the probability of each relevant category c being covered by at least one document in S. This method requires prior knowledge of the taxonomy and the learning of the probability distributions.

[12] also makes use of a cover-based definition of diversity to locate and highlight diverse concepts in documents. Given a set of *sentences* S, the *cover* of S is the union of all terms t appearing in any sentence x in them, that is:

$$Cov(S) = \bigcup_{x \in S} \bigcup_{t \in x} t$$

Assuming a function $g(i)$ that measures the benefit we have by covering a term exactly i times, the *gain* of S is:

$$Gain(S) = \sum_{i=0}^{|S|} \sum_{t \in \mathcal{T}_i} w(t) g(i)$$

where $\mathcal{T}_i$ is the set of terms appearing in exactly i sentences in S and $w(t)$ is a weight for the term t. Now, the result diversification problem is defined as follows: Given a document consisting of n sentences $\mathcal{X} = \{x_1, \ldots, x_n\}$ and an integer k, locate a set of sentences S^*, such that:

$$S^* = \underset{\substack{S \subseteq \mathcal{X} \\ |S| \leq k}}{\operatorname{argmax}} Gain(S) \tag{4}$$

3. COMBINATION OF DIVERSITY WITH OTHER CRITERIA

Diversity is most commonly used along with some other ranking criterion, most commonly that of *relevance* to the user's query. To the best of our knowledge, the first work in which the two measures were combined is [2], in which *marginal relevance*, i.e. a linear combination of relevance and diversity, is proposed as a criterion for ranking results retrieved by IR systems. A document has high marginal relevance if it is both relevant to the user query q and also exhibits minimal similarity to previously selected documents. Formally, given the set of all retrieved documents $\mathcal{X}$ and the set of already selected ones, denoted S, the document $x^* \in \mathcal{X} \backslash S$ that has the maximum marginal relevance to S is:

$$x^* = \underset{x \in \mathcal{X} \backslash S}{\operatorname{argmax}} \left[\lambda (rel(x) - (1-\lambda) \max_{x' \in S} d(x, x')) \right]$$

where $rel(x)$ is the relevance of x to the query and $\lambda \in [0,1]$. This approach has also been applied in [14] as a means to reformulate queries submitted in web search. The above formulation of the problem is called *max-sum diversification*. The objective function that is maximized this case is:

$$f(S) = (k-1) \sum_{x \in S} rel(x) + 2\lambda \sum_{x, x' \in S} d(x, x')$$

where $\lambda > 0$. Other variations of combining relevance and diversity are the *max-min diversification*, where:

$$f(S) = \min_{x \in S} rel(x) + \lambda \min_{x, x' \in S} d(x, x')$$

and also a *mono-objective* formulation of the problem in which:

$$f(S) = \sum_{x \in S} \left[rel(x) + \frac{\lambda}{|\mathcal{X} - 1|} \sum_{x' \in \mathcal{X}} d(x, x') \right]$$

[7] considers the combination of relevance and diversity and presents eight intuitive axioms that diversification systems should satisfy. However, it is shown that not all of them can be satisfied simultaneously.

The combination of these two criteria has also been studied in [18] as an optimization problem. Let once again $\mathcal{X} = \{x_1, \ldots, x_n\}$ be a set of items and D be an $n \times n$ distance matrix with the $(i,j)^{\text{th}}$ element being equal to $d(x_i, x_j)$. Let also $\mathbf{m}$ be an n-dimensional vector with the i^{th} element being equal to $rel(x_i)$. Consider, finally, an integer k and a binary n-dimensional vector $\mathbf{y}$ with the i^{th} element being equal to 1, if and only if, x_i belongs to the k most highly relevant and diverse items. Now, given a diversification factor $\lambda \in [0,1]$, we can define the problem of selecting k items that are both as relevant and diverse as possible as follows:

$$\begin{aligned} \mathbf{y}^* &= \underset{\mathbf{y}}{\operatorname{argmax}} (1-\lambda) \alpha \mathbf{y}^T D \mathbf{y} + \lambda \beta \mathbf{m}^T \mathbf{y} \\ \text{s.t.} \quad & \mathbf{1}^T \mathbf{y} = k \text{ and} \\ & y(i) \in \{0,1\}, 1 \leq i \leq n \end{aligned} \tag{5}$$

where α and β are normalization parameters.

Diversity is also combined with spatial distance, as a relevance characterazation, when solving the k-nearest diverse neighbors problem in [8].

Finally, threshold-based techniques can also be employed as in [18], where variations of the optimization problem of Equation 5 are considered (e.g. maximize the diversity of the selected items given a relevance threshold and, the dual, maximize the relevance of the selected items given a minimum required diversity). Placing a threshold on diversity however may be hard, since it requires an estimation of the achievable diversity.

4. ALGORITHMS

Given a set $\mathcal{X}$ of items, $\mathcal{X} = \{x_1, \ldots, x_n\}$, a distance metric $d(.,.)$ among items and an integer k, the diversification problem is to locate a subset S^* of $\mathcal{X}$, such that the diversity among the selected items is maximized, where the diversity of a set of

items is defined based on some specific definition of Section 2.

Generally, the diversification problem has been shown to be NP-hard. Thus, to solve large instances of the problem, we need to rely on heuristics. Many heuristic algorithms have been used in the research literature and have been employed for solving variations of the problem in more than one research fields. We can classify these heuristics into two main categories: (i) *greedy* and (ii) *interchange* (or *swap*). In following, we describe heuristics in each category and their applications.

4.1 Greedy Heuristics

The greedy heuristics are the ones most commonly used since they are intuitive and some of them are also relatively fast. Greedy heuristics generally make use of two sets: the set $\mathcal{X}$ of available items and the set S which contains the selected ones. Items are iteratively moved from $\mathcal{X}$ to S and vice versa until $|S| = k$ and $|\mathcal{X}| = n - k$. In most works, S is initialized with some item, e.g. the most relevant one, and then items are moved one-by-one from $\mathcal{X}$ to S until k of them have been selected. The item that is moved each time is the one that has the maximum *item-set distance* from S. The item-set distance, denoted $setdist(x_i, S)$, between an item x_i and a set of items S is defined based on its distance from the items in S, for example:

$$setdist(x_i, S) = \min_{x_j \in S} d(x_i, x_j)$$

or

$$setdist(x_i, S) = \frac{1}{|S|} \sum_{x_j \in S} d(x_i, x_j)$$

Ties are generally broken arbitarily.

This greedy approach is, for example, used in [20] in the context of recommender systems, where, given a set of recommendations $\mathcal{X} = \{x_1, \ldots, x_n\}$ and their degrees of relevance $rel(x_i)$, $1 \leq i \leq n$, to a user query, diverse recommendations are produced. S is initialized with the most relevant recommendation. Then, recommendations are added one-by-one to S as follows: For each recommendation x_i not yet added to S, its item-set distance from the recommendations already in S is computed. These "candidate" recommendations are then sorted in order of (i) relevance to the query and (ii) item-set distance to S. The rank of each recommendation is a linear combination of its positions in the two sorted lists. The recommendation with the minimum rank is added to S and the process is repeated until S has k recommendations. Note that the recommender system has to produce a larger number of recommendations (n) out of which the final k ones will be selected. The larger this number, the higher the possibility that more diverse recommendations will be located (at the cost of higher computation cost).

A greedy heuristic is also employed in [12] for locating diverse sentences in documents. At each round, the sentence which has the highest gain for S, as defined in Equation 4, is added to S. [1] also follows the greedy approach. In that case, an algorithm is proposed that, given the set of the top-k most relevant documents to a query, it re-orders them in a way, such that, the objective function of Equation 3 is maximized. [7] also employs another greedy variation, first presented in [9] as a solution to the p-dispersion problem, in which, at each iteration, the two remaining items with the largest pair-wise distance are added to S. A greedy solution is also used in [17] for recommenders. However, in that case, threshold values are also used to determine when two recommendations are considered distant. [8] also uses a greedy algorithm for locating the k-nearest diverse neighbors to a given item.

A special case of greedy heuristics are neighborhood heuristics. These algorithms start with a solution S containing one random item and then iteratively add items to the solution. The items to be considered at each iteration are limited based on the notion of *r-neighborhood* of an item $x_i \in \mathcal{X}$, $N(x_i, \mathcal{X}, r)$, defined as:

$$N(x_i, \mathcal{X}, r) = \{x_j \in \mathcal{X} : d(x_i, x_j) \leq r\}$$

In other words, all items that have a smaller or equal to r distance to x_i belong to its r-neighborhood. At each iteration, only items outside the r-neighborhoods of all already selected items are considered. Out of these items, one is chosen to be added to the solution. This can be the fist located item outside those r-neighborhoods, the one that has the smallest sum of distances to the already selected items or the one that has the largest sum of distances to the already selected items [6]. Note that the selection of r plays an important role as it restricts the number of items that are considered at each iteration. In fact, given a value of r, a solution S with $|S| = k$ may not even exist.

4.2 Insterchange (Swap) Heuristics

Interchange (or Swap) heuristics have also been used in the literature for solving the diversification problem. Generally, these heuristics are initialized with a random solution S and then iteratively attempt to improve that solution by interchanging an item in the solution with another item that is not in the solution. At each round, possible interchanges are the first met one that improves the solution or

the one that improves the solution the most.

An interchange heuristic that combines the relevance and diversity criteria is proposed in [17]. In this approach, S is initialized with the k most relevant items. At each iteration, the item of S that contributes the least to the diversity of the entire set, i.e. the one with the minimum item-set distance, is interchanged with the most relevant item in $\mathcal{X} \backslash S$. Interchanges stop when there are no more items in $\mathcal{X} \backslash S$ with higher relevance than a given threshold.

Another work that employs an interchange algorithm is [13], where, given a set of structured search results, the goal is to identify a subset of their features that are able to differentiate them. Starting with a random subset of features, at each iteration, one of these features is interchanged with a better candidate feature.

4.3 Other Heuristics

An algorithm for achieving diversity in database systems based on a tree index structure, i.e. the Dewey tree, is presented in [16]. Each tuple of a database relation is represented by a path in the tree. Higher levels of the tree represent more important attributes, according to the diversity ordering of the relation (see Section 2). Diverse tuples are retrieved by traversing this tree.

Motivated by the fact that the one-dimensional p-dispersion problem can be solved optimally in polynomial time, [6] considers a dimensionality-reduction heuristic that projects items in one dimension only. However, in practice, this approach does not result in good solutions.

A hybrid greedy/interchange heuristic is used in [4] in the context of continuous data. In this case, a diverse subset S is located using a greedy approach and then its diversity is further improved by performing interchanges.

Another related approach is that of [11], where, given the set of a database query results, these results are grouped in k clusters and the corresponding k medoids are retrieved as a subset of k representative and diverse results.

Finally, in [18], where the diversification problem is formulated as an optimization one, a solution is approximated via optimization techniques that include problem relaxation and quantization.

5. EVALUATION MEASURES

The diversity of a set S of selected items can be evaluated by the value of the objective function $f(S)$ based on which the diversity problem is defined, e.g. Equation 1. This approach is used in most of the related work (e.g. [20, 17, 5, 12, 18]). The computed value can be normalized by the corresponding value for the set S^*, i.e. the optimal solution to the diversification problem. This, however, is not always feasible due to the high cost of computing the optimal solution.

In the field of IR systems, there has been an effort to adapt traditional IR evaluation measures so as to become diversity-aware. A key difference of these approaches is that the retrieved results are usually viewed as an *ordered list* instead of a set. These adapted measures are usually applied along with novelty-based or coverage-based diversity definitions.

For example, [3] proposes evaluating retrieved results through a weighted *Normalized Discounted Cumulative Gain Measure* (denoted α-NDCG), a measure often used in the context of IR systems that measures the gain of an item being at a specific position of the list given the items that precede it. Given an ordered list of items, the k^{th} element of the list's gain vector, denoted $\mathbf{G}$, is computed based on Equation 2 as:

$$\mathbf{G}[k] = \sum_{i=1}^{m} J(x_k, o_i)(1-\alpha)^{r_{o_i,k-1}}$$

and the corresponding cumulative gain vector, denoted $\mathbf{GC}$, is computed as:

$$\mathbf{CG}[k] = \sum_{j=1}^{k} \mathbf{G}[j]$$

Usually, the elements of the cumulative gain vector are weighted according to their position in the list, so the discounted cumulative gain vector, denoted $\mathbf{DGC}$, is computed as:

$$\mathbf{DCG}[k] = \sum_{j=1}^{k} \frac{\mathbf{G}[j]}{\log_2(1+j)}$$

The discounted cumulative gain vector computed for a list is finally normalized by the ideal discounted cumulative gain. However, the computation of this is an NP-complete problem and, in practice, its value is approximated via heuristics.

The adaptation of the NDCG measure is also considered in [1], where NDCG is aggregated over all available categories that a document may be related to (see Section 2). This variation is called *Intent-Aware Normalized Discounted Cumulative Gain Measure* (denoted NDCG-IA). Its value for the k^{th} element of a list S of items retrieved for a query q is:

$$NDCG\text{-}IA(S,k) = \sum_{c} P(c|q) NDCG(S,k|c)$$

The same aggregation method can be applied to other IR measures as well, such as *Mean Reciprocal Rank* (MRR) and *Mean Average Precision* (MAP).

Fianlly, a redundancy-aware variation of the traditional *precision* and *recall* measures is considered in [19]:

$$Redundancy\text{-}Precision = \frac{R^-}{R^- + N^-}$$

and

$$Redundancy\text{-}Recall = \frac{R^-}{R^- + R^+}$$

where R^- is the set of non-delivered redundant documents, N^- is the set of non-delivered non-redundant ones and R^+ is the set of delivered redundant ones.

Besides deriving appropriate measures, user studies are also central in evaluating the usefulness of diversification. In a recent study, two thousand volunteers from the BookCrossing[2] community were asked to rate recommendations produced by using diversification techniques [20]. The results vary according to the method used to acquire the initial recommendations, but overall users rated the diversified recommendations higher than the non-diversified ones in all cases, as long as diversity contributed up to 40% to the linear combination of the relevance and diversity measures. A higher contribution led to a lower overall rating by the users. An interesting finding is that, when diversified results were presented to the users, the individual recommendations where generally rated lower but the overall rating of the recommendation list as a whole was higher.

6. CONCLUSIONS

In this work, we presented the various definitions of the result diversification problem proposed in the research literature and classified them into three main categories, namely content-based, novelty-based and cover- age-based. These three factors are closely related and, therefore, most related work considers more than one of them. We also reviewed different approaches taken for the combination of diversity with other ranking criteria, most commonly that of relevance, to the user's information need. We classified the algorithms used in the literature for locating diverse items into two main categories (greedy and interchange) and also discussed other used approaches. Finally, we showed how diversity is evaluated.

7. REFERENCES

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *WSDM*, pages 5–14, 2009.

[2] J. G. Carbonell and J. Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *SIGIR*, pages 335–336, 1998.

[3] C. L. A. Clarke, M. Kolla, G. V. Cormack, O. Vechtomova, A. Ashkan, S. Büttcher, and I. MacKinnon. Novelty and diversity in information retrieval evaluation. In *SIGIR*, pages 659–666, 2008.

[4] M. Drosou and E. Pitoura. Diversity over continuous data. *IEEE Data Eng. Bull.*, 32(4):49–56, 2009.

[5] M. Drosou, K. Stefanidis, and E. Pitoura. Preference-aware publish/subscribe delivery with diversity. In *DEBS*, 2009.

[6] E. Erkut, Y. Ülküsal, and O. Yeniçerioglu. A comparison of -dispersion heuristics. *Computers & OR*, 21(10):1103–1113, 1994.

[7] S. Gollapudi and A. Sharma. An axiomatic approach for result diversification. In *WWW*, pages 381–390, 2009.

[8] J. R. Haritsa. The KNDN problem: A quest for unity in diversity. *IEEE Data Eng. Bull.*, 32(4):15–22, 2009.

[9] R. Hassin, S. Rubinstein, and A. Tamir. Approximation algorithms for maximum dispersion. *Operations Research Letters*, 21(3):133 – 137, 1997.

[10] G. Koutrika and Y. E. Ioannidis. Personalized queries under a generalized preference model. In *ICDE*, pages 841–852, 2005.

[11] B. Liu and H. V. Jagadish. Using trees to depict a forest. *PVLDB*, 2(1):133–144, 2009.

[12] K. Liu, E. Terzi, and T. Grandison. Highlighting diverse concepts in documents. In *SDM*, pages 545–556, 2009.

[13] Z. Liu, P. Sun, and Y. Chen. Structured search result differentiation. *PVLDB*, 2(1):313–324, 2009.

[14] F. Radlinski and S. T. Dumais. Improving personalized web search using result diversification. In *SIGIR*, pages 691–692, 2006.

[15] K. Stefanidis, M. Drosou, and E. Pitoura. PerK: personalized keyword search in relational databases through preferences. In *EDBT*, pages 585–596, 2010.

[16] E. Vee, U. Srivastava, J. Shanmugasundaram, P. Bhat, and S. Amer-Yahia. Efficient computation of diverse query results. In *ICDE*, pages 228–236, 2008.

[17] C. Yu, L. V. S. Lakshmanan, and S. Amer-Yahia. It takes variety to make a world: diversification in recommender systems. In *EDBT*, pages 368–378, 2009.

[18] M. Zhang and N. Hurley. Avoiding monotony: improving the diversity of recommendation lists. In *RecSys*, pages 123–130, 2008.

[19] Y. Zhang, J. P. Callan, and T. P. Minka. Novelty and redundancy detection in adaptive filtering. In *SIGIR*, pages 81–88, 2002.

[20] C.-N. Ziegler, S. M. McNee, J. A. Konstan, and G. Lausen. Improving recommendation lists through topic diversification. In *WWW*, pages 22–32, 2005.

[2] http://www.bookcrossing.com

SmartCIS: Integrating Digital and Physical Environments*

Mengmeng Liu Svilen R. Mihaylov Zhuowei Bao Marie Jacob
Zachary G. Ives Boon Thau Loo Sudipto Guha

Computer & Information Science Department, University of Pennsylvania, Philadelphia, PA, USA
{mengmeng,svilen,zhuowei,majacob,zives,boonloo,sudipto}@cis.upenn.edu

ABSTRACT

With the increasing adoption of networked sensors, a new class of applications is emerging that combines data from the "digital world" with real-time sensor readings, in order to intelligently manage physical environments and systems (e.g., "smart" buildings, power grids, data centers). This leads to new challenges in providing programmability, performance, *extensibility*, and *multi-purpose* heterogeneous data acquisition. The ASPEN project addresses these challenges by extending data integration techniques to the distributed stream world, and adding new abstractions for physical phenomena. We describe the architecture and implementation of our ASPEN system and its showcase intelligent building application, SmartCIS, which was demonstrated at SIGMOD 2009. We summarize the new query processing algorithms we have developed for integrating highly distributed stream data sources, both in low-power sensor devices and traditional PCs and servers; describe query optimization techniques for federations of stream processors; and detail new capabilities such as incremental maintenance of recursive views. Our algorithms and techniques generalize across a wide range of data from RFID and light measurements to real-time machine usage monitoring, energy consumption and recursive query computation.

1. INTRODUCTION

Low-cost networked sensors are resulting in a new class of applications that combine data from the "digital world" with sensor readings, to create environments that intelligently manage resources and assist humans. Examples include intelligent power grids [19], smart hospitals [18], home health monitors, energy-efficient data centers, and building visitor guides. In such applications, there is a need to bring together disparate data from databases (e.g., site information, patient treatments, maps) with data from the Web (e.g., weather forecasts, calendars), from streaming data sources (e.g., resource consumption within a server), and from sensors embedded within an environment (e.g., generator temperature, RFID readings, energy levels) — in order to support decision making by high-level application logic. Today this sort of data integration, if done at all, is performed by a proprietary software stack over fixed devices.

In order for intelligent environments to reach their potential, what is necessary is an *extensible*, *multi-purpose* data acquisition and integration substrate through which the application can acquire data — without having to be coded with special support for new device or network types Over the past 30 years, the database community has developed a wealth of techniques for performing data integration through views and related formalisms [11]. Likewise, declarative queries have been shown to be useful beyond databases, with extensions for distributed data stream management [2, 3, 4, 9] and sensor networks [5, 6, 14]. The key question is how to develop a unified declarative query and integration substrate, which supports a multitude of stream and static data sources on heterogeneous, possibly unreliable networks. Computation should be expressed in a single query language and "pushed" to where it is most appropriate, taking into account capabilities, battery life, rates of change, and network bandwidth.

The ASPEN (Abstraction-based Sensor Programming ENvironment) project tackles these issues, extending the formalisms of data integration (schema mappings, views, queries) to the distributed stream world. We are developing (1) new query processing algorithms suitable for integrating highly distributed stream data sources, both in low-power sensor devices [15, 16] and more traditional PCs and servers [13], (2) query optimization techniques for federations of stream processors specialized for sensor, wide area, and LAN settings, and (3) new datatypes, query extensions, and data description language abstractions for environmental monitoring and for routing information to users. In support of smart environments, we seek a *single data access layer* for integrating sensor, stream, and database data, regardless of origins. This single programming interface over heterogeneous sensors and stream sources distinguishes us

*This work was funded by NSF III IIS-0713267, NOSS CNS-0721541, and a grant from Lockheed Martin.

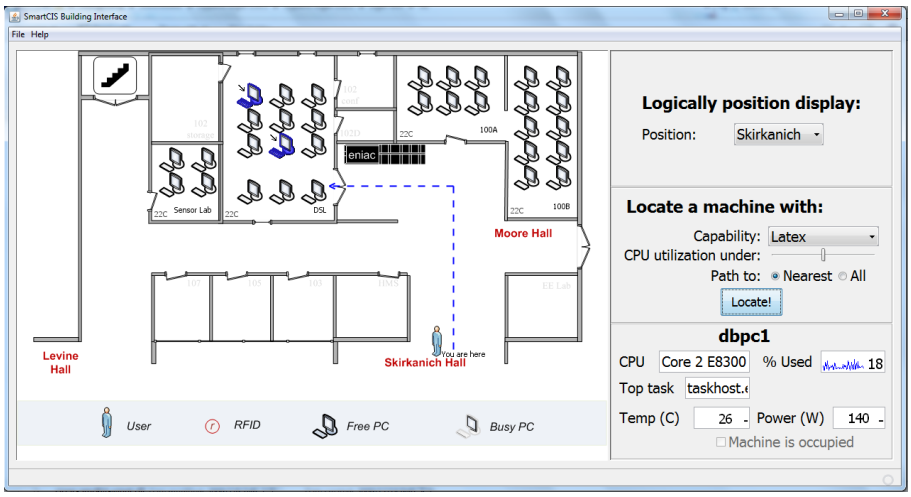


Figure 1: Display indicating a path to, and information about, the nearest machine with LaTeX.

from other sensor systems [7, 10, 14].

The showcase application for the ASPEN architecture, which we term SmartCIS, involves instrumenting Penn's Computer and Information Science (CIS) Department buildings, labs, and data centers to help improve energy efficiency, guide visitors to their desired destinations, and locate resources. Our live demonstration of SmartCIS [12] at SIGMOD 2009 received Honorable Mention for Best Demo. SmartCIS consists of GUI and query logic built over the ASPEN data integration substrate. It combines information from on-site sensors (e.g., pressure-sensitive seat cushions, RFID tags, energy meters) with data from the Web (calendars) and from our Distributed Systems Laboratory at Penn (machine and desk-occupied status; machine configurations).

We describe the SmartCIS prototype in Section 2. Then we present the underlying ASPEN system: its architecture (Section 3), federated query optimizer (Section 4), distributed stream query processor (Section 5), and sensor network subsystem (Section 6). We summarize related work in Section 7 and conclude in Section 8.

2. SMARTCIS BUILDING APPLICATION

One of the most compelling emerging applications of sensors are intelligent building environments: they promise to make the experience of visiting a large building or a hospital less disorienting, to make buildings or large data centers more energy-efficient, to help occupants remember to take their medications or make it to a next meeting. A distinguishing feature of such environments, versus other sensor network applications, is a need to bring together database data with streaming data from the Web or Internet and streaming data from sensor devices. The task of designing a smart building can be separated into three tiers: data acquisition and integration, query and control logic, and a user-interface view (analogous to model-view-controller architectures).

The initial version of SmartCIS focuses on monitoring and querying the data of interest to CIS students and faculties, as well as system administrators: lab status, machine activity, resource consumption, and machine physical state. We target two main tasks: giving a real-time update of the building state, and guiding students to the resources they need. Through the SmartCIS GUI, visitors can see occupied and unoccupied desks in the laboratories and on-site (detected through the seat sensors); their positions in the building (obtained via RFID); temperature, light, and energy usage levels for every machine and lab; room reservation status from Google Calendar; and the resources available at each machine (e.g., software, special equipment). Visitors can see status information or issue a query for directions (a physical path) to a machine with a particular resource.

2.1 User Experience

SmartCIS interacts with users through a touch interface on a kiosk or (for the demo) a tablet PC. Figure 1 shows a screen shot of our graphical interface, which centers around a building schematic. In the full application, the user will see the individual information on a kiosk located somewhere in the building. Our screen shot shows the demo application, which has a selector in the upper right-hand corner enabling a SIGMOD attendee to choose a simulated kiosk location.

Buildings, entrances and exits, rooms, and machines are illustrated schematically. Their status is refreshed in real-time based on data streams from the environment and the Web, combined with database information about locations and configurations. Rooms are grayed out when marked as reserved in a standard Google calendar, or when their lights are out (as detected by sensors). Machines are grayed out when they are currently in use (as detected by high CPU utilization or a pressure-sensitive seat cushion connected to a Crossbow iMote). The presence of a user is detected through active RFID tags (IRIS motes that broadcast a low-power signal that

is tracked by stationary motes located throughout the building hallways) and is indicated in the schematic.

The user can also trigger new continuous queries over the streaming data in the system. Clicking on a machine icon switches the right-hand pane to show details about that device: its host name (from a database table mapping coordinates to machine identities), CPU type (also from the database), CPU utilization and the most CPU-intensive task (from a "soft sensor" application), temperature (from an iMote), and energy (from a USB energy meter or an IP-based Power Distribution Unit or PDU). A double-click opens up a secondary window showing energy consumption on a per-task basis (scaling overall energy consumption by the amount of resources consumed by each process). Finally, a visitor can also request to be directed to an available machine with specific resources (e.g., software packages like Microsoft Office or a video editor). A shortest-path query is initiated between the user's current location and the nearest available room with the specified resource.

2.2 Sensors and Data Sources

The data sources underpinning SmartCIS are heterogeneous, requiring a variety of *wrappers* (interface modules), and can be divided into four broad categories.

Sensor devices. We use Crossbow IRIS and iMote2 sensors to monitor the rooms' and workstations' temperatures, as well as light levels (useful for determining if a lab is open). A pressure-sensitive seat cushion attached to a wireless mote monitors whether someone is seated at each desk in the lab. A "wrapper" periodically extracts this value and sends it along a data stream. Energy meters are physically plugged into machines and feed raw readings into the system. To track users' locations, "mote" sensors are embedded in the hallways at major intersection points, at approximately every 50 feet. These sensors listen for a "beacon" transmission from an active RFID device (also a mote) carried by an occupant and based on the strength of the signal determine where that person is positioned in the building.

"Soft" sensors. Servers and workstations run daemon software to monitor machine activity: jobs executing, users logged in, CPU utilization, number of requests being handled in a Web server application, etc. In addition, the status of ASPEN, our back-end data acquisition and integration substrate itself, is also monitored: the queries and plans being executed, the counts of tuples received and sent for every operator, etc. This helps developers diagnose problems at the query execution level and also helps determine per-query energy usage.

Web and streaming data sources. A wrapper periodically polls a Google Calendar for room reservations. Another wrapper polls energy usage from a Web interface to our lab's power distribution units (PDUs).

Databases. A conventional DBMS stores the coordinates of each RFID detector (the motes have no built-in absolute positioning capability), a list of machine configurations and locations, and a table of "routing points" describing possible path segments and distances in the building in order to suggest routes to resources.

The data from these inputs is "hooked" to the SmartCIS GUI through a series of Stream SQL queries and view definitions, plus callbacks to Java functions that update the graphical widgets. It is trivial to extend the GUI to support visual or auditory alarms if machines exceed a temperature or load factor, or to aggregate the sensor data across users, applications, or machines. Even the path routing in the GUI is done declaratively, using recursive extensions to Stream SQL. We next describe how SmartCIS maps onto the ASPEN substrate that provides distributed Stream SQL services.

3. SYSTEM ARCHITECTURE

The SmartCIS system consists of three major components: the graphical interface described previously, which can be deployed on kiosks; the ASPEN data integration and acquisition substrate, which includes two query runtime systems (one that enables certain computations to be "pushed" to sensor devices, and one that does distributed stream processing over PC-style servers) plus a federated query optimizer; and wrappers and interfaces over the actual sensors, databases, and machines. (See Figure 2.) Components of the ASPEN substrate appear in boldface. (Ultimately ASPEN will also include support for schema mappings and query reformulation, but SmartCIS does not require these components.)

Most of the research innovations are in the ASPEN modules. ASPEN takes a query (Stream SQL with extensions for devices and for routing query output to displays) and invokes a federated query optimizer that partitions it into two portions (see Figure 2): a subquery that is "pushed" out to the sensor network and sensor devices, and the remaining computations that get executed on our distributed stream engine for servers.

The distributed sensor engine, whose core features were described in [15], is novel in supporting not only aggregation and selection queries over sensor devices, but also *in-network* joins between devices. This is useful in SmartCIS, for instance, when we return machine temperature data for workstations that are in use. We detect that a workstation is being used by checking the status of the seat cushion as well as the light level at an adjacent chair. The most efficient query strategy is to perform a proximity-based join between status of seat cushion and light level sensors (with a threshold applied on the light level), and route the temperature information across the sensor network only if the light level threshold is not met. A query optimizer decides where to perform the join computation on a sensor-by-sensor basis.

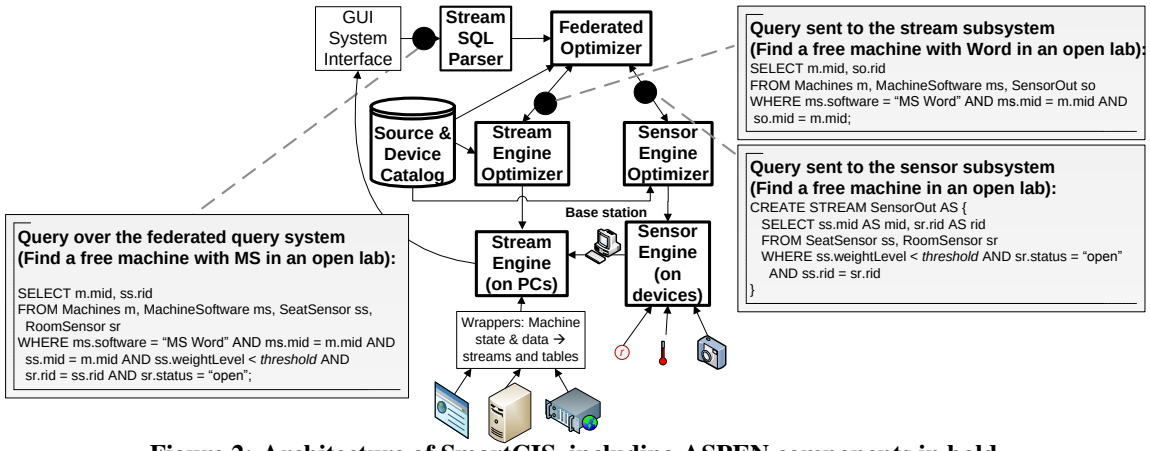


Figure 2: Architecture of SmartCIS, including ASPEN components in bold.

Our distributed stream engine, described in [13, 20], supports not only Stream SQL queries over windowed data, but also *transitive closure queries* to compute neighborhoods and paths. The stream engine brings together streaming data, database data, and the data returned by the subqueries sent to the sensor engine. It is also responsible for computing suggested routes for building occupants to get to their destination: this can be done in real-time based on the occupant's current position and information about the topology of the buildings (connected by routing points described previously).

4. FEDERATED OPTIMIZER

ASPEN's federated query optimizer assigns an incoming query across multiple subsystems, each of which has its own custom optimizer and cost metric, customized to the target device and network capabilities (e.g., energy, latency, bandwidth). We give an example of the federated optimizer's optimization in SmartCIS.

Suppose we have two types of sensors deployed in the lab, seat sensors and room sensors. Each seat sensor is pre-initialized with information about its position relative to a machine and the room; it reports the occupied-status of the seat cushion to which it is attached. Each room sensor is pre-initialized with its room, and detects the current light level to tell whether the room is occupied or not. Suppose we also have a static table $Machines$ storing machine information for the lab, and a dynamic stream $MachineSoftware$ containing information about installed software and versions from a web page. The user may pose a query to find all the free machines in an open lab which have "Word."

```
SELECT m.mid, sr.rid
FROM Machines m, MachineSoftware ms,
  SeatSensor ss, RoomSensor sr
WHERE software = "Word" AND ms.mid =
  m.mid AND ss.mid = m.mid AND ss.weightLevel
  < threshold AND sr.rid = ss.rid AND
  sr.status = "open";
```

There are multiple plausible ways of splitting the query. One method pushes the `SeatSensor-RoomSensor` join and all relevant selection conditions to the sensor subsystem, then sends the output to the stream engine. (Example SQL for this scenario is shown in Figure 2.) Alternatively, we can issue *two* subqueries to the sensor subsystem: one to fetch `SeatSensor` readings above threshold, and the other to fetch `RoomSensor` readings with open status. Intuitively, the first query partitioning is likely to return fewer results to the stream system only if the predicates are selective.

The federated optimizer must choose among these and other plans by minimizing an over-arching cost metric (e.g., query latency). This metric may be *different* from the metrics of the "local" optimizers for the underlying stream and sensor engines (e.g., bandwidth, energy consumption). The federated optimizer must find a query partitioning that, when each subquery is optimized according to its target platform's specific metric, results in the best plan with respect to the federated optimizer's over-arching metric. Its plan enumeration strategy resembles that of [8], which predicts the query plan produced by an external optimizer, in order to produce the minimum-cost plan according to its own metric.

5. STREAM ENGINE

Our stream engine is derived from the distributed SQL processor from ORCHESTRA [20]. This engine supports horizontal partitioning of data across nodes within a cluster or peer-to-peer network, and is based on a push-style query processing model. We enhance the engine with support for continuous queries (where the query is active unless deliberately stopped) over windows, where the size of the sliding window tells the system when to evict expired tuples. The engine can seamlessly combine data from streaming sources, tables partitioned throughout the cluster, ODBC/JDBC sources, and the sensor query engine. The query optimizer uses a Volcano-style top-down dynamic programming algorithm, and takes into account the network latency as well as data transmission rate when estimating the cost of a certain query plan.

A novel aspect of our engine is its support for *re-*

cursive queries (such as shortest paths) computed (and incrementally maintained) over streaming data. Such queries commonly appear in sensor settings. We have developed techniques, documented in [13], based on (1) the use of a particular kind of *data provenance* that enables us to detect when a tuple in the output stream should be expired, (2) early pruning of intermediate results that do not contribute to the output, and (3) careful use of buffering to reduce traffic. SmartCIS exploits these features to compute path queries, when a user requests to be directed to a resource within the building.

6. SENSOR ENGINE

The sensor engine collects environmental data, potentially from several independent sensor networks. Each sensor network deployment consists of wireless devices situated within the observed environment, and a gateway node to the core of the ASPEN system. One of the sensor devices is connected via USB to the gateway node, serving as a *base station* for the rest of the wireless network. We do not assume that all wireless nodes remain in the radio range of the base station; we focus on effectively utilizing multi-hop wireless networks.

Sensor query capabilities. Our sensor subsystem supports windowed Stream SQL queries (subqueries sent by the federated optimizer) with arbitrary selection conditions, and optionally a single in-network join. Selection and join predicates can include not only standard comparisons and Boolean operations, but also arithmetic operators and several utility functions (e.g., hash, random value). We model each mote sensor network deployment as a single relation with attributes including sensor values (e.g., temperature, light, humidity, battery level, RFID being detected, ADC values) and *soft-state* readings (e.g., memory available, local time). Not all attributes need to be defined for every node, as our system allows for different device capabilities. We also allow for additional data values to be stored at each device from external tables. The update rate for physical sensors is specified as part of the query, as are other parameters such as query start times, join window size, etc.

Basic operation and coordination. The gateway machine not only bridges between TCP and ZigBee (mote) networks, but also plays a supervisory role in the sensor engine. It supervises the construction and initialization of the wireless network after all wireless devices are turned on (described in the next sub-section). It also collects statistical information for the federated optimizer, such as the number of wireless nodes present, network diameter and distribution of values in the different regions of the network. The gateway also serves as a coordinator for *partitioning and storing* certain tables within the sensor network: often it is useful to take certain database tables (e.g., a mapping between node identity and position or role) and to partition them such that one tuple is stored at each wireless node. Then we can push selection conditions relating to these tables directly into the network, optimizing communication efficiency. Finally, it takes sensor network subqueries from the federated optimizer, and performs a "local" network-specific optimization of those queries for the sensor network.

We now briefly describe initialization, optimization, and query execution; [15, 16] provide more details.

Sensor network initialization. Right after the wireless network is started, the first step is to create a sensor network topology that approximates the connectivity graph among sensors. A set of spanning trees is created, one rooted at the base station node, and others rooted at nodes located at opposite extremities of the network. Each internal node in the trees maintains a *summary* (Bloom filter, histogram, R-tree) of the values for a particular attribute that appear in each subtree. The summaries are used for content-based routing [15].

Query pre-optimization. When a subquery is sent to the gateway machine, a query pre-processor first separates the predicates in the query into selections and joins. Then, predicates from each group are separated into static and dynamic subgroups, depending of their attributes being exclusively static or not. Each static join predicate is further fed into a pattern matcher, which, given a collection of summaries built on various static attributes, decides whether the predicate is suitable for content routing using our substrate. In essence, the pattern matcher identifies those join predicates usable for routing, versus runtime evaluation. Finally, the gateway node considers different ways of distributing the evaluation of expressions. Consider the following query with user-defined functions F and G:

```
SELECT S.u+S.v, F(2*S.u+S.v), G(S.u), G(S.v)
FROM SENSORS S [windowsize=1
   sampleinterval=100]
WHERE S.u > 0 AND S.id = 0;
```

This query asks for four evaluations on attributes for every node in the network satisfying $S.u > 0 \wedge S.id = 0$. If at a given node, the selection condition is satisfied, a trivial execution strategy will compute the four evaluations and send them to the base station node. A more efficient strategy will send only $S.u$ and $S.v$, which the base station can in turn use to compute the four original evaluations. In general the intermediate evaluations can themselves be expressions. The problem is exponential in the number of evaluations, and our implementation uses a combination of dynamic programming and heuristics, which performs optimally for queries we used in our testing and experimentation.

Once the pre-optimization is finished, the query is encoded and flooded to every node in the wireless network.

Distributed optimization and execution. For single-

relation queries there is no optimization phase, so query execution proceeds immediately. Otherwise, query optimization is invoked at each node, which checks if its attributes satisfy the static selection conditions. If so it initiates a *directed flood* routing request, searching for nodes that mutually satisfy the static join selection conditions. The directed flooding algorithm uses the summaries constructed during initialization [15]. If such nodes are discovered, their selection conditions are checked, and a *join pair* is established. Consider the query:

```
SELECT S.id, T.id, S.time
FROM SENSORS S, SENSORS T
     [windowsize=3 sampleinterval=100]
WHERE S.id < 25 AND hash(S.u) % 2 = 0
  AND T.id > 50 AND hash(T.u) % 2 = 0
  AND T.y = S.x + 5 AND S.u = T.u
```

First, all nodes (playing the role of S) check if their id is less than 25. If so, they issue a routing request to find nodes for which $T.y = S.x + 5$ and $T.id > 50$. Following the evaluation of the static predicates and establishment of paths between joining nodes, a *join node* is assigned for each join pair. Candidate join nodes are those on the path connecting the source nodes, as well as the base station. The join node is chosen using a cost model, based on the estimated relative selectivities of each relation, and the relative network distances (see [16]).

Query execution samples attributes (humidity, temperature, light levels, ADC voltages) at regular intervals, and evaluates the dynamic selection conditions. If the conditions are satisfied, a tuple containing intermediate evaluations is sent to either the join node (if applicable) or the base station. Each join node collects tuples from both relations and computes the join result by evaluating the dynamic join predicate. The join node also tracks any changes to the relative selectivity of the relations it handles, and may trigger adaptation of the join node placement, as described in [16]. Finally, results are sent to the base station and then to the gateway machine.

7. RELATED WORK

In the past decade several influential distributed stream systems [3, 9, 17] have been proposed. These have established the basic semantics and query languages for stream processing. In parallel, stream SQL techniques have been shown to be highly advantageous in a sensor setting [5, 6, 14]. Work such as REED [1] has shown that there is promise in coupling the two classes of systems. We seek to take this idea further, with a federated model supporting distributed optimization across multiple cooperating stream sub-systems, each tailored to particular device classes; support for data integration capabilities; recursion for path and region queries.

8. CONCLUSIONS

This paper provided a technical overview of the SmartCIS "smart building" application and its underlying ASPEN substrate. We introduced new query processing schemes for integrating highly distributed stream data sources, both for low-power sensor devices and servers, as well as query optimization techniques for federations of stream processors. Future work includes designing a more flexible federated optimizer, adaptive query processing techniques for our highly distributed setting, and support for user-defined functions.

9. REFERENCES

[1] D. J. Abadi, W. Lindner, S. Madden, and J. Schuler. An integration framework for sensor networks and data stream management systems. In *VLDB*, pages 1361–1364, 2004.

[2] A. Arasu, S. Babu, and J. Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB J.*, 15(2), 2006.

[3] M. Balazinska, H. Balakrishnan, S. Madden, and M. Stonebraker. Fault-tolerance in the Borealis distributed stream processing system. *ACM Trans. Database Syst.*, 33(1), 2008.

[4] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *CIDR*, 2003.

[5] A. J. Demers, J. Gehrke, R. Rajaraman, A. Trigoni, and Y. Yao. The Cougar project: a work-in-progress report. *SIGMOD Record*, 32(3), 2003.

[6] A. Deshpande and S. Madden. MauveDB: Supporting model-based user views in database systems. In *SIGMOD*, 2006.

[7] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, F. Reiss, S. Rizvi, E. Wu, O. Cooper, A. Edakkunni, and W. Hong. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, 2005.

[8] L. M. Haas, D. Kossmann, E. L. Wimmers, and J. Yang. Optimizing queries across diverse data sources. In *VLDB*, 1997.

[9] R. Huebsch, J. M. Hellerstein, N. Lanham, B. T. Loo, S. Shenker, and I. Stoica. Quering the Internet with PIER. In *VLDB*, 2003.

[10] N. Khoussainova, E. Welbourne, M. Balazinska, G. Borriello, G. Cole, J. Letchner, Y. Li, C. Ré, D. Suciu, and J. Walke. A demonstration of cascadia through a digital diary application. In *SIGMOD*, New York, NY, USA, 2008.

[11] M. Lenzerini. Tutorial - data integration: A theoretical perspective. In *PODS*, 2002.

[12] M. Liu, S. Mihaylov, Z. Bao, M. Jacob, Z. G. Ives, and B. T. Loo. SmartCIS: Integrating digital and physical environments. In *SIGMOD*, 2009.

[13] M. Liu, N. E. Taylor, W. Zhou, Z. G. Ives, and B. T. Loo. Maintaining recursive views of regions and connectivity in networks. *TKDE*, 2010. Special issue on best papers of ICDE 2009.

[14] S. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong. Design of an acquisitional query processor for sensor networks. In *SIGMOD*, 2003.

[15] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. A substrate for in-network sensor data integration. In *DMSN*, August 2008.

[16] S. R. Mihaylov, M. Jacob, Z. G. Ives, and S. Guha. Dynamic join optimization in multihop wireless sensor networks. In *Proc VLDB*, 2010. To appear.

[17] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, resource management, and approximation in a data stream management system. In *CIDR*, 2003.

[18] J. V. Sutherland, W.-J. van den Heuvel, T. Ganous, M. M. Burton, and A. Kumar. *Future of Intelligent and Extelligent Health Environment*, volume 118/2005, pages 278–312. IOS Press, 2005.

[19] J. Taft. The intelligent power grid. *Innovating for Transformation: The Energy and Utilities Project*, 6:74–76, 2006. Available from www.utilitiesproject.com.

[20] N. E. Taylor and Z. G. Ives. Reliable storage and querying for collaborative data sharing systems. In *ICDE*, 2010.

Report on the EDBT/ICDT 2010 Workshop on Updates in XML

Michael Benedikt
Oxford University, UK
michael.benedikt@gmail.com

Daniela Florescu
Oracle Corporation, USA
dflorescu@mac.com

Philippa Gardner
London Imperial College, UK
p.gardner@imperial.ac.uk

Giovanna Guerrini
University of Genova, Italy
guerrini@disi.unige.it

Marco Mesiti
University of Milano, Italy
mesiti@dico.unimi.it

Emmanuel Waller
University of Paris Sud, France
waller@lri.fr

ABSTRACT

The first international workshop on *Updates in XML* [1] was held in conjunction with the EDBT/ICDT conference in Lausanne (Switzerland) on March 22, 2010, and attracted approximately 25 participants, culminating with about 40 attending the last session. This paper summarizes the main ideas presented in the workshop as well as interesting perspectives identified by the participants.

1. OUTLINE

The theme of the workshop was updates in any data model, with an emphasis on XML and recent models. Updates have always been considered in databases, as change is inherent to their lifecycle and that of their applications. Updates arise in many different contexts and situations, and bring up numerous issues and problems, many of which having strong practical impact. Over years, a powerful collection of approaches, techniques and algorithms has been developed. However, many problems remain open. New issues arise in recent data models like XML, semi-structured, graph-based, RDF and probabilistic among others, some widely used in practice, and updates have recently known a new gain of interest. The goal of the workshop was thus to address open problems in general and new issues in recent models, by bringing together academics, practitioners, users and vendors. It was also to stimulate discussions on existing results, possibly compared w.r.t. new issues, and on the connections between the different topics in update management, as well as on future trends.

In response to the call for papers, 11 high quality submissions were received. Each paper was carefully reviewed by at least three members of the program committee and external reviewers. As a result of this process, 6 papers have been selected for presentation. In addition, Michael Benedikt, Daniela Florescu and Philippa Gardner accepted the invitation to give invited talks. The accepted papers cover a large variety of both practical and theoretical topics on updates, in XML and other models such as RDF, probabilistic XML, and relational, ranging from dynamic labelling schemes and schema evolution, to view updates, and updates and functional dependencies. In what follows, we first present the main ideas and issues proposed by the invited speakers and then the papers selected by the program committee. Finally, discussions arise during the workshop and concluding remarks are presented. The slides of workshop talks can be found on the workshop web page (`http://updates2010.lri.fr/`).

2. INVITED TALKS

The first invited talk *Static Analysis of Declarative Update Languages* by Michael Benedikt (based on joint work with James Cheney) started the workshop. Declarative XML update languages are harder to analyze than queries. Static type inference and type checking are certainly more difficult, and even basic effect (i.e., what parts of a document does an update impact) analysis problems are complex. A survey of previous results on analysis of XML updates, and their relation to problems in XPath/XQuery, is provided. The focus then moves on the query/update interaction problem: do an update and a query interact? This problem lies at the core of many optimization problems, like view maintenance under declarative updates and minimization of number of passes in update evaluation.

The query/update interaction problem is particularly interesting in that it requires re-examining the notion of query provenance. What does it mean precisely for a query to read, or depend on, one portion of the document? In the second part of the talk a framework for describing the dependence of

a query on a document in terms of updates is presented. The framework makes sense for any data model, but instantiated for XML it gives an approach to update interaction problems. The basic theory, and then the specifics of the implementation for a fragment of the W3C XQuery Update Facility are discussed.

The second invited paper *Reasoning about Client-side Programming* was by Philippa Gardner (based on joint work with Gareth Smith, Mark Wheelhouse, Adam Wright and Uri Zarfaty). A formal, compositional specification of the Document Object Model (DOM), a W3C XML Update library, has been presented in PODS 2008, concentrating on Featherweight DOM, a fragment of DOM that focuses on the XML tree structure and simple text nodes. Since the formal reasoning is compositional, working with a minimal set of commands, a complete reasoning for straight-line code can be obtained and invariant properties of simple DOM programs can be verified.

The work is based on a recent breakthrough in program verification, based on analysing a program's use of resource. The idea is that the reasoning should follow the programmers' intuitions about which part of the computer memory the program touches. This style of reasoning was introduced by O'Hearn (Queen Mary) and Reynolds (CMU) in their work on Separation Logic for reasoning modularly about large C-programs (e.g., Microsoft device driver code, Linux). In this work, the range of local resource reasoning is substantially extended, introducing Context Logic to reason about programs that directly manipulate complex data structures such as XML.

In the talk, an overview of the theoretical and practical work on reasoning about DOM is given, highlighting recent developments which include: *(i)* handling of DOM Core Level 1; *(ii)* reasoning about the combination of JavaScript and DOM to provide, for example, secure mashups for a more flexible, secure integration of outsourced payment services; *(iii)* on-going work on a verification tool for automatically reasoning about DOM programs and the identification of key examples of web applications on which to test DOM reasoning. An ultimate challenge is to develop the necessary reasoning technology to provide a safe and secure web environment on which to build the next generation of web applications.

Dana Florescu presented the last invited paper *General Ranting about XML (Reasoning about Updates)*. Dana started with an industrial prospective on XML and XQuery management and pointed out that XML is nowadays pervasive (*"part of the DNA of computing"*) and that different reasons motivate the use of XML in very diverse contexts, in which the requirements for update mechanisms are different. XQuery is then briefly discussed, highlighting that, despite the name, it is a computationally complete functional programming language. The main languages involved in updates for XML are then introduced: XQuery Update Facility (XQUF for short), scripting extensions of XQUF expressions, and Pending Update Lists (PULs) obtained by evaluating XQUF. These languages are already known, studied, implemented, and of acceptable quality, thus the position is that no new language should be invented. Moreover, in reasoning about updates and analysing their properties, subsetting of the languages should be avoided, though considering the whole languages one may get only sufficient conditions.

The talk then focused on four different application contexts in which updates are crucial and the ability to reason about them would be greatly beneficial: *execution in the cloud*, *disconnected execution*, *transactional models* and *XML time machine*. In the cloud, updates travel on the network, and arrive on different machines, where they are put in queues, with no information on when they are actually applied. Problems may arise because some updates are not yet performed at some point, or because of the order in which they are actually performed. The notion of consistency between updates should thus be changed into a notion of *eventual consistency*. Concerning disconnected execution, a first point is that XML applications should follow a one-tier architecture and everything should be written in a single language, e.g., XQuery (or an extension). Indeed, many languages imply many translations (and optimization), which imply many data transformations, which in turn imply many data transfers. In such a context, the same piece of code can be executed on the client or on the server (code mobility) and the need for reasoning about updates emerges from this disconnected execution of XQuery on the client. For what concerns transactions, classical ACID properties and lock based mechanisms are not adequate for all the diverse contexts in which XML documents are employed. A more flexible definition of conflicts and checkin/checkout approaches *a-la SVN* to merge updates from different users are more appropriate in many contexts. A new generic transaction model is needed, since different applications may want different definitions and behavior of transactions. Finally, XML time machine refers to the ability of keeping all the document versions, together with

the operations that generated them. This entails seeing the PULs as data and storing and querying them.

These applications would benefit from algorithms to reason about updates and to analyse them. Both static and dynamic analysis are relevant. Specific reasoning that would be useful are updates minimization, aggregation, inverse computation, commutativity analysis, detection of inconsistencies and constraint (and schema) violations.

3. REFEREED PAPERS

Hicham Idabal presented the paper *Regular Tree Patterns: A Uniform Formalism for Update Queries and Functional Dependencies in XML*, by Hicham Idabal and Françoise Gire. Given an XML functional dependency *fd* and a class of updates $\mathcal{U}$, *fd* is said to be independent with respect to $\mathcal{U}$ if and only if any XML document satisfies *fd* after any update q of $\mathcal{U}$, provided that it did it before q. The paper focuses on the following problem: is it possible to detect if an XML functional dependency *fd* is independent with respect to a class of updates $\mathcal{U}$? This problem is addressed when both the functional dependency and the class of updates are specified with regular tree patterns. The use of regular tree patterns federates most of the known approaches for expressing XML functional dependencies while allowing to capture some constraints not expressible so far. The addressed problem is in general PSPACE-hard, but a sufficient condition testable in polynomial time is exhibited, ensuring the independence of a functional dependency with respect to a class of updates.

Benoît Groz presented the paper *The View Update Problem for XML* by Slawek Staworko, Iovka Boneva and Benoît Groz. The paper addresses the problem of update propagation across views in the setting where both the view and the source database are XML documents. A simple class of XML views that remove selected parts of the source document is considered. The considered update operations permit to insert and delete subtrees of the document. The focus of the approach is on constructing propagations that are *(i)* schema compliant, i.e., when applied to the source document they give a document that satisfies the document schema; *(ii)* side-effect free, i.e., the view of the new source document is exactly as the result of applying the user update to the old view. A special structure allowing to capture all such propagations is presented, and how to use this structure to capture only those propagations that affect minimally the parts of the document which are not visible in the view is shown. Finally, a general outline of a polynomial algorithm constructing a unique propagation is presented.

Federico Cavalieri presented his paper *EXup: An Engine for the Evolution of XML Schemas and Associated Documents*. XML Schema is employed for describing the type and structure of XML documents. Schema evolution means that a schema is modified and the effects of the modification on instances are faced. XSUpdate is a language that allows to easily identify parts of an XML Schema, apply a modification primitive on them and finally define an adaptation for associated documents, while *EXup* is the corresponding engine for processing schema modification and document adaptations. This paper presents an engine for the evaluation of XSUpdate statements against XML Schemas and associated documents. The presented engine relies on the translation of XSUpdate statements in XQuery Update expressions.

Evgeny Kharlamov presented the paper *Updating Probabilistic XML* by Evgeny Kharlamov, Werner Nutt and Pierre Senellart. The paper investigates the complexity of performing updates on probabilistic XML data for various classes of probabilistic XML documents of different succinctness. Two elementary kinds of updates are considered, insertions and deletions, that are defined with the help of a locator query that specifies the nodes where the update is to be performed. For insertions, two semantics are considered, depending on whether a node is to be inserted once or for every match of the query. Deterministic updates over probabilistic XML is first discussed, and then the algorithms and complexity bounds are extended to probabilistic updates. In addition to a number of intractability results, the main result is an efficient algorithm for insertions defined with branching-free queries over probabilistic models with local dependencies. Finally, the problem of updating probabilistic XML databases with continuous probability distributions is discussed.

Martin F. O'Connor presented the paper *Desirable Properties for XML Update Mechanisms* by Martin F. O'Connor and Mark Roantree. Many approaches have been proposed for processing queries efficiently. The ever-increasing deployment of XML in industry and the real-world requirement to support efficient updates to XML documents has more recently prompted research in dynamic XML labelling schemes. In this paper, an overview of the recent research in dynamic XML labelling schemes is provided. The motivation is to define a set of properties that represent a more holistic dynamic labelling scheme and to present authors' findings

through an evaluation matrix for most of the existing schemes that provide update functionalities.

Matthias Hert presented the paper *Updating Relational Data via SPARQL/Update* by Matthias Hert, Gerald Reif and Harald Gall. The semantics of the data is not explicitly encoded in the relational model, but implicitly on the application level. Ontologies and Semantic Web technologies provide explicit semantics that allows data to be shared and reused across application, enterprise, and community boundaries. Converting relational data to RDF is often not feasible, therefore an ontology-based access to relational databases is proposed. While existing approaches focus on read-only access, the proposed OntoAccess approach adds ontology-based write access to relational data. OntoAccess consists of the update-aware RDB to RDF mapping language R3M and algorithms for translating SPARQL/Update operations to SQL. The paper presents the mapping language, the translation algorithms, and a prototype implementation of OntoAccess.

4. DISCUSSION AND CONCLUSIONS

A large part of the workshop has been devoted to issues related in some way to update reasoning – analysis, dependability, propagation – issues at different levels. Specifically, there have been talks on update-query interaction, on update-constraint interaction, on propagation of updates across views and on propagation of updates on schema to the corresponding documents. Though the levels at which the problems are investigated are different (different classes of updates are considered) as well as the employed formalisms, the issues faced by various approaches have many similarities, thus the workshop has been beneficial in giving the opportunity to more closely relate approaches that have many contact points. Moreover, the invited talks allow to broaden the picture and to identify the different dimensions and alternatives that emerge in reasoning about updates. Other issues that emerged relate to more expressive models (probabilistic XML and SPARQL), and to efficient update support.

The final discussion mainly covered two other important issues in XML updates. The first one is benchmarking. Though the need for a benchmark for XML updates is commonly felt, the very diverse characteristics of XML document collections and their different update requirements lead to the conclusion that a single benchmark is not enough. The second one is the role of schemas. XML documents may come with or without a schema, the schema may as well come later, that is, be inferred from documents. Schema information may be useful to better organize document storage so as to make operations on documents more efficient, however, their instability and dynamism introduce further problems that should be faced.

5. ACKNOWLEDGMENTS

We would like to thank the program committee members and the external reviewers for their efforts in the realization of this workshop and workshop participants. A special thanks to the Laboratoire de Recherche en Informatique (LRI), University of Paris South at Orsay, for providing a grant as support for the organization of the event. We wish to express our gratitude to the EDBT/ICDT Organization and the General Chair, Stefano Spaccapietra, for their support in the workshop preparation.

6. REFERENCES

[1] F. Daniel, L. M. L. Delcambre, F. Fotouhi, I. Garrigós, G. Guerrini, J.-N. Mazón, M. Mesiti, S. Müller-Feuerstein, J. Trujillo, T. M. Truta, B. Volz, E. Waller, L. Xiong, E. Zimányi: Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland. ISBN 978-1-60558-990-9, ACM 2010.

Report on the First International Workshop on Cloud Data Management (CloudDB 2009)

Xiaofeng Meng[1] Jiaheng Lu[1] Jie Qiu[2] Ying Chen[2] Haixun Wang[3]
{xfmeng,jiahenglu}@ruc.edu.cn, {qiujie, yingch}@cn.ibm.com, haixunw@microsoft.com
[1]School of Information and DEKE, MOE, Renmin University of China, Beijing China
[2]IBM Research - China
[3]Microsoft Research Asia, Beijing

1. INTRODUCTION

The first ACM international workshop on cloud data management was held in Hong Kong, China on November 6, 2009 and co-located with the ACM 18th Conference on Information and Knowledge Management (CIKM). The main objective of the workshop was to address the challenge of large data management based on cloud computing infrastructure. The workshop brings together researchers and practitioners in cloud computing and data-intensive system design, programming, parallel algorithms, data management, scientific applications and information-based applications interested in maximizing performance, reducing cost and enlarging the scale of their endeavors.

The workshop attracted 11 submissions from Asia, Canada, Europe and the United States, out of which the program committee finally accepted 5 full papers and 3 short papers. The accepted papers focused on cloud-based indexing and query processing, cloud platform availability, cloud replication and system development.

2. KEYNOTE PRESENTATION

The keynote speech, titled "EMC Decho: A Real World Use Case of Cloud Computing" was delivered by Bill Sun, Engineer Manager in EMC Center of Excellence China. He presented EMC's perspective about cloud computing, and shared some of their experience in building a reliable infrastructure to provide personal cloud services for millions of users. Bill Sun highlighted some of challenges they are facing in building robust and reliable infrastructures and described their potential remedies.

The first challenge is the increasing importance that personal information has to the eyes of the users. As a simple example, the first picture of one's newborn son is a digital artifact that a user is likely to want to preserve through one or several generations. The value of digital information to users has become such important that preserving one's digital data cannot be tied to a particular device or application. The information saved in these devices are more valuable than devices and should live much longer.

The second challenge is to search effectively in various devices. Current-day technologies for searching personal archives of digital data still have strong limitations. For example, how can you easily find your pictures in Las Vegas taken in July last year? How can you find a presentation you got from a colleague six months back about a particular project? Most of time, we have to organize the information by five "C": i.e. Context, Content, Calendar, Coordinates and Contacts. Effective management of such personal information with five "C" is a big challenge.

The major conclusion is that personal cloud is what they believe the most effective approach to address those challenges in personal information management, where all the information are saved in a secure well managed cloud storage system. Decho will work as the center or the hub to synchronize all users' information across multiple devices through the personal cloud, including PC, cell phone, or even NetBook.

3. RESEARCH PAPERS

The technical paper session consisted of eight presentations, whose main points are summarized next. Together, they give a glimpse to the exciting new developments spurred by data management in the cloud. These papers cover a variety of topics. We believe that these papers will provide researchers and developers with a brief glimpse into this exciting new technology, specifically from the perspective of cloud data management.

The paper entitled *Personalization as a Service: Architecture and Case Study* focuses on how to provide personalized services for individual users in the cloud environment. H. Guo, J. Chen, W. Wu and W. Wang first analyzed the main issues and challenges of using the traditional server-side user profiles for personalized services in the cloud. Then they presented the architecture of Personalization as a Service (PaaS) in which the client-side user modeling method is employed to support personalized cloud services. The

main idea is to decouple user modeling components from cloud services by observing the user's interactions on all of their cloud client devices collectively. As a result, user model can be shared across cloud services and used in a pay-as-you-go way. The client-side user modeling avoids the server overhead and provides unique user experiences with minimal user intervention. They finally give a case study of a personalized cloud search service solution according to the PaaS architecture.

X. Zhang, J. Ai, Z. Wang, J. Lu and X. Meng proposed an efficient approach to build a multi-dimensional index for cloud computing systems in the paper *"An Efficient Multi-Dimensional Index for Cloud Data Management"*. Their approaches can process typical multi-dimensional queries including point queries and range queries efficiently. Besides, frequent change of data on big amount of machines makes the index maintenance a challenging problem. To cope with this problem they proposed a cost estimation-based index update strategy that can effectively update the index structure. They describe experiments showing that their indexing techniques improve query efficiency by an order of magnitude compared with alternative approaches, and scale well with the size of the data. Their approach is quite general and independent from the underlying infrastructure and can be easily carried over for implementation on various cloud computing platforms.

The topic considered in *Packing the Most Onto Your Cloud* by A. Aboulnaga, Z. Wang and Z. Zhang is one particular optimization problem, namely scheduling sets of Map-Reduce jobs on a cluster of machines (a computing cloud). They present a scheduler that takes job characteristics into account and finds a schedule that minimizes the total completion time of the set of jobs. Their scheduler decides on the number of cluster nodes to assign to each job, and it tries to pack as many jobs on the machines as the machine resources can support. To enable flexible scheduling and packing of jobs onto machines, they run the Map-Reduce jobs in virtual machines, although their scheduling approach can be applied in any Map-Reduce scheduler. Their scheduling problem is formulated as a constrained optimization problem, and they experimentally demonstrate using the Hadoop open source Map-Reduce implementation that the solution to this problem results in benefits up to 30%.

Query Processing of Massive Trajectory Data based on MapReduce is addressed by Q. Ma, B. Yang, W. Qian and A. Zhou. Traditional trajectory data partitioning, indexing, and query processing technologies are extended so that they may fully utilize the highly parallel processing power of large-scale clusters. They also showed that the append-only scheme of MapReduce storage model can be a nice base for handling updates of moving objects. Preliminary experiments show that this framework scales well in terms of the size of trajectory data set. The limitation of traditional trajectory data processing techniques and their future research direction are also discussed.

The approach considered in *Leveraging a Scalable Row Store to Build a Distributed Text Index* by N. Li, J. Rao, E. Shekita and S. Tata is a distributed text index called HIndex, by judiciously exploiting the control layer of HBase, which is an open source implementation of Google's Bigtable. Such leverage enables them to inherit the good properties of availability, elasticity and load balancing in HBase. They also present the design, implementation, and a performance evaluation of HIndex.

F. Wang, J. Qiu, J. Yang, B. Dong, X. Li, and Y. Li proposed a metadata replication based solution to enable Hadoop high availability by removing single points of failures in Hadoop in the paper titled *Hadoop High Availability through Metadata Replication*. Single points of failures mean that the whole system becomes out of work due to the failure of critical nodes where only a single copy of data exists. The solution involves three major phases. In the initialization phase, each standby/slave node is registered to active/primary node and its initial metadata (such as version file and file system image) are caught up with those of active/primary node. In the replication phase, the runtime metadata (such as outstanding operations and lease states) for fail-over in future are replicated. Finally, in the fail-over phase, standby/new elected primary node takes over all communications. The solution presents several unique features for Hadoop, such as runtime configurable synchronization mode. The experiments demonstrate the feasibility and efficiency of their solution.

In the Paper entitled *How Replicated Data Management in the Cloud can benefit from a Data Grid Protocol - the Re:GRIDiT Approach*, L. Voicu and H. Schuldt developed, implemented and evaluated the Re:GRIDiT protocol for managing data in the grid. Re:GRIDiT provides support for concurrent access to replicated at different sites without any global component and supports the dynamic deployment of replicas. Since it has been designed independent from any underlying grid middle-ware, it can be seamlessly transferred to other environments like the cloud. They present the Re:GRIDiT protocol, show its applicability for cloud data management, and provide performance results of the evaluation of the protocol in realistic cloud settings.

The topic in *The Design of Distributed Real-time Video Analytic System* by T. Yu, B. Zhou, Q. Li, R. Liu, W. Wang and C. Chang is to propose a distributed scalable infrastructure VAP (Video Analytic Platform) for supporting real-time video stream analysis. In VAP, the application requirements are represented as a Directed Acyclic Graph (DAG), where nodes stand for video analysis computation modules and links show data flow and dependencies between nodes. VAP leverages UIMA (Unstructured Information Management Architecture) framework as the data flow control engine and multiple commodity databases as the storage and computation resources. The actual executions of video analysis computation modules have been pushed down into database engine to minimize the data movement cost.

4. CONCLUSION

CloudDB 2009 was the first CIKM-associated workshop addressing the challenges of large database services based on the cloud computing infrastructure. Whilst these emerging services have reduced the cost of data storage and delivery by several orders of magnitude, there is significant complexity involved in ensuring large data service can scale when one needs to ensure consistent and reliable operation under peak loads. Cloud-based environment has the technical requirement to manage data center virtualization, lower cost and boost reliability by consolidating systems on the cloud.

A first conclusion that can be drawn from this workshop is that the cloud systems should be geographically dispersed to reduce their vulnerability due to earthquakes and other catastrophes, which increase technical challenge on a great level of distributed data interpretability and mobility. Data interoperability is even more essential in the future as one component of a multi-faceted approach to many applications.

A final conclusion is that existing research works in the area of cloud-based data management are still somehow immature and significant room for progress exists. The works presented in the workshop mainly focused on adapting existing Grid and Map/Reduce techniques to the cloud environment. The participants agreed that many open challenges still remain such as cloud data security and the efficiency of query processing in the cloud. The participants also expressed interest in the organization of a conference dedicated to the issues raised by data management in the cloud.

5. ACKNOWLEDGEMENT

We would like to thank the program committee members, keynote speakers, authors and attendees, for making CloudDB 2009 a successful workshop. Jiaheng Lu was supported by 863 National High-Tech Research Plan of China (No: 2009AA01Z133). Finally, we also express our great appreciation for the support from Renmin University of China and IBM research-China.

CALL FOR PARTICIPATION

DMSN 2010: Seventh International Workshop on Data Management for Sensor Networks

September 13, 2010, Grand Copthorne Waterfront Hotel, Singapore
(in conjunction with VLDB 2010)
http://www.cs.ucy.ac.cy/~dmsn10/

Sponsored by the Cooperating Objects Network Of Excellence (CONET)

Workshop Aim

The scope of the workshop includes all important aspects of sensor data management, including data acquisition, processing, and storage in remote wireless networks; the handling of uncertain sensor data; and the management of heterogeneous and sometimes sensitive sensor data in databases.

Detailed Program

08:00 - 09:00: Registration (Pre-registration: Sunday, September 12th, between 5 pm to 8 pm)

09:00 - 09:15: Opening Remarks

09:15 - 10:30: Session I: Keynote by Prof. Kian-Lee Tan (National Univ. of Singapore, Singapore)
Keynote Title: *"What's NExT? Sensor + Cloud!?"*

10:30 - 11:00: Coffee Break

11:00 - 12:30: Session II: Data Provenance and Query Processing

- *"Provenance-based Trustworthiness Assessment in Sensor Networks"*, Hyo-Sang Lim (Purdue, USA), Yang-Sae Moon (Kangwon National Univ., South Korea), Elisa Bertino (Purdue, USA)
- *"Facilitating Fine Grained Data Provenance using Temporal Data Model",* Mohammad R. Huq, Andreas Wombacher and Peter M. G. Apers (Univ. of Twente, Netherlands)
- *"Processing Strategies for Nested Complex Sequence Pattern Queries over Event Streams"*, Mo Liu, Medhabi Ray, Elke A. Rundensteiner, Daniel J. Dougherty (Worcester Polytechnic Institute, USA), Chetan Gupta, Song Wang (HP Labs, USA), Ismail Ari (Ozyegin Univ., Turkey), Abhay Mehta (HP Labs, USA)

12:30 - 14:00: Lunch Break (Kiwi Lounge: Level 2)

14:00 - 15:30: Session III: Mobile Sensor Networks and Outlier Detection

- *"Query Driven Data Collection and Data Forwarding in Intermittently Connected Mobile Sensor Networks"*, Wei Wu (NUS, Singapore), Hock Beng Lim (Nanyang Technological Univ.) and Kian-Lee Tan (NUS, Singapore)
- *"DEMS: A Data Mining Based Technique to Handle Missing Data in Mobile Sensor Network Applications"*, Le Gruenwald, Md. Shiblee Sadik, Rahul Shukla and Hanqing Yang (Univ. of Oklahoma, USA)
- *"PAO: Power-Efficient Attribution of Outliers in Wireless Sensor Networks"*, Nikos Giatrakos (Univ. of Piraeus, Greece), Yannis Kotidis (AUEB, Greece), Antonios Deligiannakis (Technical Univ. of Crete, Greece)

15:30 - 16:00: Coffee Break

16:00 - 17:45: Session IV: Panel *"Future Directions in Sensor Data Management: A Panel Discussion"*
Panel Moderator: Demetris Zeinalipour (Univ. of Cyprus, Cyprus)
Panelists: Yanlei Diao (Univ. of Massachusetts - Amherst, USA), Le Gruenwald (NSF, USA), Christian S. Jensen (Aarhus Univ., Denmark) and Kian-Lee Tan (NUS, Singapore)

17:45 - 18:00: Closing Remarks

Program Chairs:

Wang-Chien Lee (Pennsylvania State University, USA)
Demetris Zeinalipour (University of Cyprus, Cyprus)

4th Extremely Large Databases Conference

October 6-7, 2010

SLAC National Accelerator Laboratory
Menlo Park, California

- Complex Analytics at Extreme Scale
- Lessons Learned, Practical Solutions and Emerging Trends
- Perspectives from Science, Industry, and Academia

Discussed Topics Include:

- Complex Scientific and Industrial Analytics at Extreme Scale
- Operational Issues with Managing Large Data Clusters
- Behind the Scenes of Big Science Projects
- Existing Scientific Tools and Formats
- Emerging Technologies for Complex Extreme Scale Analytics
- Science Benchmark
- Extreme Scale Architectures and New Hardware Trends
- Data Preservation and Integration
- Automated Information Extraction, Content Curation and Machine Learning

Early registration (until July 31): $100
Late registration (after July 31): $150
Registration will be closed when capacity is reached

http://xldb.org/4