A First Step Towards Stream Reasoning

Emanuele Della Valle, Stefano Ceri, Davide Francesco Barbieri, Daniele Braga, and Alessandro Campi

Dip. di Elettronica e Informazione, Politecnico di Milano, Milano, Italy
{name.surname}@polimi.it

Abstract. While reasoners are year after year scaling up in the classical, time invariant domain of ontological knowledge, reasoning upon rapidly changing information has been neglected or forgotten. On the contrary, processing of data streams has been largely investigated and specialized Stream Database Management Systems exist. In this paper, by coupling reasoners with powerful, reactive, throughput-efficient stream management systems, we introduce the concept of Stream Reasoning. We expect future realization of such concept to have high impact on the future Internet because it enables reasoning in real time, at a throughput and with a reactivity not obtained in previous works.

Keywords: Data Streams, Reasoning, Real-time, Throughput-efficiency, Urban Computing, Pervasive Computing.

1 Introduction and Motivation

Semantics is more and more evoked as a powerful tool to facilitate interoperability, flexibility and adaptability. The growing scalability of reasoning techniques [1] is key to the relevant role that semantics will play in the future Internet. While reasoners scale up in the classical, static domain of ontological knowledge, reasoning upon rapidly changing information has been neglected or forgotten.

Data streams are unbounded sequences of time-varying data elements; they occur in a variety of modern applications, such as network monitoring, traffic engineering, sensor networks, RFID tags applications, telecom call records, financial applications, Web logs, click-streams, etc. Processing of data streams has been largely investigated in the last decade [2], specialized Data Stream Management Systems (DSMSs) have been developed, and features of DSMSs are becoming supported by major database products, such as Oracle and DB2.

The combination of reasoning techniques with data streams gives rise to **Stream Reasoning**, an unexplored, yet high impact, research area. To understand the potential impact of Stream Reasoning, we can consider the emblematic case of Urban Computing [3,4,5,6] (i.e., the application of pervasive computing to urban environments). The very nature of Urban Computing can be explained by means of data streams, representing real objects that are monitored at given locations: cars, trains, crowds, ambulances, parking spaces, and so on. Reasoning about such streams can be very effective in reducing costs: for instance, looking

J. Domingue, D. Fensel, and P. Traverso (Eds.): FIS 2008, LNCS 5468, pp. 72–81, 2009.

for parking lots in large cities may cost up to 40% of the daily fuel consumption. Problems dramatically increase when big events, involving lots of people, take place; a typical Urban Computing problem is to help citizens willing to participate to such events in finding a parking lot and reaching the event locations in time, while globally limiting the occurrences of traffic congestions.

Some years ago, due to the lack of data, solving Urban Computing problems looked like a Sci-Fi idea. Nowadays, a large amount of the required information can be made available on the Internet at almost no cost: computerized systems contain maps with the commercial activities and meeting places (e.g., Google Earth), events scheduled in the city and their locations, positions and speed information of public transportation vehicles and of mobile phone users [5], parking availabilities in specific parking areas, and so on. However, current technologies are not up to the challenge of solving Urban Computing problems: this requires combining a huge amount of static knowledge about the city (i.e., urbanistic, social and cultural knowledge) with an even larger set of data streams (originating in real time from heterogeneous and noisy data sources) and reasoning above the resulting time-varying knowledge.

A new generation of reasoners is clearly needed in order to simultaneously instruct the car GPS of numerous citizens with the fastest route to the most convenient parking lot during exceptional events. Time constraints for such a reasoner are very demanding (i.e., few ms per query) because citizens are continuously making driving decisions and the traffic keeps evolving; therefore, continuous inference is required. In this work, we define Stream Reasoning as a new paradigm, based upon the state of the art in DSMS and reasoning, in order to enable such applications. By coupling reasoners with powerful, reactive, throughput-efficient stream management systems, we expect to enable reasoning in real time, at a throughput and with a reactivity not obtained in previous works.

In the rest of the paper, we identify the problem we want to untangle (Section 2). We introduce a Conceptual Architecture for Stream Reasoning (Section 3) that instantiates the pluggable algorithmic framework proposed in the LarKC project [7,8]. We present two stream reasoning frameworks based on such architecture, one representing an evolutionary approach that combines existing solutions (Section 4), the other representing a revolutionary approach that proposes a new reasoning paradigm (Section 5). We conclude the paper discussing the challenges we are facing while planning our future work (Section 6).

2 Problem Definition

Our attempt to combine data stream and reasoning technologies starts from terminology. Database (DB) and Knowledge Engineering (KE) communities often use different terms to indicate the same concepts. DB community distinguishes among schema and data, whereas KE community distinguishes among factual, terminological, and nomological knowledge. The notion of data is close to the notion of factual knowledge, and similarly the notion of schema is close to the notion of terminological knowledge. Nomological knowledge is information about

rules defining actions and action-types and governing means-ends relationships in a given culture or society (e.g., when it rains, traffic gets slower); this notion is somehow captured by constraint languages for DBs, but it is mainly peculiar of KE. For the purpose of this paper, we name "*knowledge*" both terminological and nomological knowledge (thus we include in term knowledge the DB notion of schema) and we use "*data*" as a synonymous of factual knowledge.

Knowledge and data can change over time. For instance, in Urban Computing, names of streets, landmarks, kinds of events, etc. change very slowly, whereas the number of cars that go through a traffic detector in five minutes changes very fast. In order to classify knowledge and data according to the frequency of their changes we first need to introduce the notion of "*observation period*", defined as the period when we the system is subject to querying. In the context of this paper, we consider knowledge as **invariable during the observation period**; only data can change. Of course, knowledge is subject to change, but then the mutating part of the system is not object of observation. This is not surprising: in the DB context, change of schemas occur by means of create or alter table command; while, for instance, the alter table command is executed all query processing relative to that table is suspended.

Examples of invariable knowledge, in the case of Urban Computing, include obvious terminological knowledge (such as an address is made up by a street name, a civic number, a city name, and a ZIP code), which defines the *conceptual schema* of the application, and less obvious nomological knowledge that describes how the world is expected to be (e.g., given traffic lights are switched off or certain streets are closed during the night) or to evolve (e.g., traffic jams appears more often when it rains or when important sport events take place).

Data can be further classified according with the frequency they are expected to change.

1. **Invariable** data: data that do not change in the observation period, e.g. the names and lengths of the roads.
2. **Periodically changing** data, for which a temporal law describing their evolution is present in the invariable knowledge. We can distinguish:
   (a) *Probabilistic* data, e.g. the fact that a traffic jam is present in the west side of Milan due to bad weather or due to a soccer match is taking place in San Siro stadium;
   (b) *Pure periodic* data, e.g. the fact that every night at 10pm Milan west-side overpass road closes.
3. **Event driven changing** data that got updated as a consequence of some external event not described in the knowledge, which are further characterized by the mean time between changes:
   (a) *Fast*, as an example consider the intensity of traffic (as monitored by sensors) for each street in a city;
   (b) *Medium*, as an example consider roads closed for accidents or congestion due to traffic;
   (c) *Slow*, as an example consider roads closed for scheduled works.

Traditional databases are suitable for capturing relatively small quantity of knowledge in their schema and huge dataset of both invariable data and event driven changing data whose mean time between changes is slow or medium. Periodically changing data can be modeled by means of triggers that perform updates; for example a trigger may update the state of a traffic light when it gets switch off for night-time.

Current reasoners are suitable for capturing large and complex knowledge, but at the cost of small datasets. Complex form of periodically changing data can be modeled by means of rules. However, reasoners cannot capture event-driven changing data whose mean time between changes is fast.

If we consider dynamic query generation, we observe that reasoners are best equipped to execute in reaction to the user's invocation, while many modern applications such as urban computing (but also network monitoring, financial analysis, sensor networks, etc.) require long-running, or continuous, queries or reasoning tasks.

Stream Database Management Systems (DSMS) represent a paradigm change in the database world because they move from persistent relations to transient streams, with the innovative assumption that streams can be *consumed* on the flight (rather then stored forever) and from user-invoked queries to *continuous queries*, i.e., queries which are persistently monitoring streams and are able to produce their answers even in the absence of invocation. DSMSs can support parallel query answering over data originating in real time and can cope with burst of data by adapting their behavior and gracefully degradating answer accuracy by introducing higher approximations.

Is combining data stream and reasoning possible? Can the innovation so far confined within the DB community be leveraged in realizing a new generation of reasoners able to cope with continuous reasoning tasks?

3 A Conceptual Architecture for Stream Reasoning

We are developing the Stream Reasoning vision with the LarKC European Research Project[1]. LarKC proposes [7,8] a pluggable algorithmic framework which will be implemented on a distributed computational platform. The pluggable algorithmic framework ideally includes five steps to be iterated until a good enough answer [9] is found:

1. *retrieve* relevant resource/content/context;
2. *abstract* by extracting information, calculating statistics and transforming to logic,
3. *select* relevant problems/methods/data,
4. *reason* upon the aggregated knowledge, and
5. *decide* if a new iteration is needed.

In Figure 1 we present our vision in plugging data stream technologies in the LarKC framework. The top part of the figure represents the problem space

[1] http://www.larkc.eu

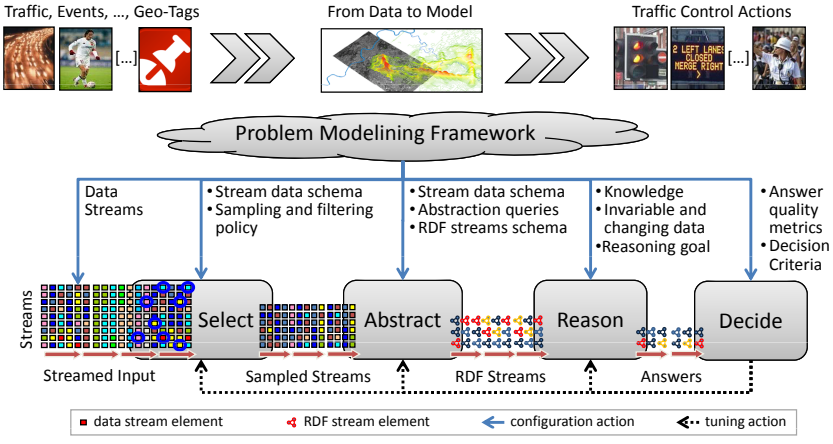

Fig. 1. Conceptual System Architecture

grounded in the Urban Computing field: data from the urban environment (e.g., traffic info, events, geo-tags, etc.) are translated in models and by reasoning on those models traffic control actions can be taken (e.g., controlling traffic lights, showing messages on traffic information panels, asking police intervention, etc.). The bottom part of the figure represents the four pluggable steps of the LarKC approach that we consider for Stream Reasoning[2] interconnected by the data that flow from left to right.

Data arrives to the Stream Reasoner as streamed input. A first step **selects** the relevant data in the input stream by exploiting *load-shedding* techniques [10]. Such techniques were developed to deal with bursty streams that may have unpredictable peaks during which the load may exceed available system resources. The key idea behind load-shedding is to introduce sampling policies that probabilistically drop stream elements as they are input to the selection step. Sampling and filtering policies can be either a) specified explicitly at stream-registration time, or b) inferred by gathering statistics over time, or c) by explicitly including punctuation in streams [11].

An example of sampling and filtering policy could be: if in a city a data stream originates from each traffic control camera, images should be sampled at given times rather than be continuously analyzed; in normal traffic condition, each stream could be sampled every 5 minutes, with options for increasing or decreasing the sampling rates (in congestion condition sample every 2 minutes, at night sample every 10 minutes).

The sampled streams resulting from the selection step are fed into a second step that **abstracts** from fine grain data streams into aggregated events. Such abstraction step can be done either by exploiting data compression techniques or

[2] We are explicitly omitting the retrieval step, because data stream retrieval should not be different from any other resource retrieval, therefore we will relay on pluggable components conceived by others.

by aggregation queries. Data compression techniques includes the usage of histograms [12] or wave-lets [13], when the abstraction is meant to be an aggregation (e.g., counting the number of cars running through traffic detectors), and using Bloom filters [14] for duplicate elimination, set difference, or set intersection.

By **abstraction query** we mean, a continuous query that, given a large set of (possibly) unrelated low-level data in the input streams, produces an aggregated event. For instance, the abstraction step may rise a traffic congestion alert for a given street if the number of cars counted by the traffic control camera exceed 100 cars and it has been continuously increasing in the last 15 minutes.

The main proposition brought up in this paper is that, either for doing the abstraction step, or immediately after the abstraction, data streams are consolidated as **RDF streams**. RDF streams are new data formats set at the confluence of conventional data streams and of conventional atoms usually injected into reasoners. At this stage of our research, we envision two alternative formats for RDF streams:

- A **RDF molecules stream** is an unbounded bag of pairs $< \rho, \tau >$, where ρ is a RDF molecule [15] and τ is the timestamp that denotes the logical arrival time of RDF molecule ρ on the stream;
- A **RDF statements stream** is a special case of RDF molecules stream in which ρ is an RDF statement instead of an RDF molecule.

Descending from the two formats, we conceive two different stream reasoning frameworks. RDF molecule streams introduce stream reasoning as a progressive process, allowing for reuse of existing DSMS and reasoners. RDF statements stream introduce stream reasoning as a revolutionary process, requiring upon reasoners the same paradigm shift as the introduction of data streams upon databases. Section 4 and 5 describe respectively the two frameworks.

As last step, before producing the solution of the application problem of our concern (e.g., a congestion situation is monitored and traffic is rerouted according to planning activities), the answering process reaches the **decision** step. In such step quality metrics and decision criteria, defined by the application developer, are used to check if the quality of the answer is good enough and to adapt the behavior of each step (e.g., changing the sampling frequency).

4 RDF Molecules Stream Reasoning Framework

As we have just stated, RDF molecule streams introduce stream reasoning as a progressive process. They allow for reuse of existing DSMS and reasoners by coupling them using a transcoder and a pre-reasoner (see Figure 2).

In particular, the abstraction step can be realized using a DSMS and a transcoder. The DSMS receives the sampled data streams and generates an abstracted data stream by continuously answering the abstraction queries designed by the application developer, which typically perform an aggregation of events. The **transcoder** generates a stream element $< \rho, \tau >$, where ρ is a RDF molecule and the timestap τ typically corresponds to the end of the aggregation interval, and puts it in the outgoing RDF stream.

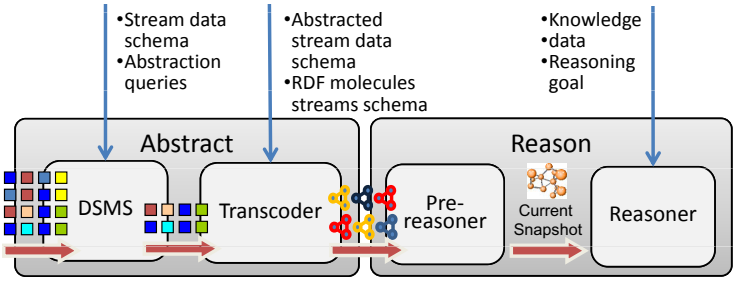

Fig. 2. RDF Molecules Stream Reasoning Framework

We choose RDF molecules [15] as the minimum amount of information, because RDF molecules are the finest component into which an RDF graph can be decomposed without loss of information. Given that a data stream is composed by tuples and each tuple carries a minimum amount of processable information, a direct transcoding of each tuple into an RDF molecule is always possible.

For instance, in our Urban Computing example we may have a system of traffic sensors that feed a data stream by recording every sensed car across a given road. An aggregator associated with each sensor counts the number of vehicles, distinguishing them according to their type; then, the transcoder encodes this information into an RDF molecule stream element. For instance, an RDF molecule for this example is composed of four triples connected by a blank node `_:x`.

$$\left\langle \begin{array}{l} \text{http://uc.ex/tcc\#123} \quad \text{uc:measure} \quad \text{\_:x.} \\ \text{\_:x} \quad \text{uc:numberOfCars} \quad 120. \\ \text{\_:x} \quad \text{uc:numberOfTrucks} \quad 70. \\ \text{\_:x} \quad \text{uc:numberOfOtherVehicles} \quad 37. \end{array} , Jun.17, 09:06:16AM \right\rangle$$

RDF molecule streams are fed into **pre-reasoners** that perform the incremental maintenance of materialized *RDF snapshots*, i.e. RDF views describing the state of the system at a given time, which are given as input to reasoners according to application-specific strategies. Reasoners are not aware of time and produce a set of answers that remain valid until pre-reasoners produce the next snapshot. The efficient incremental materialization of RDF snapshots performed by pre-reasoners is a research challenge under investigation; background studies concern the incremental maintenance of materialized ontologies [16] and indexing of temporal XML documents [17].

5 RDF Statements Stream Reasoning Framework

As we anticipated in Section 3, we are also considering a more revolutionary approach, where streams are directly represented in RDF, and therefore continuous and/or aggregation queries can be directly expressed in RDF languages. Compared to RDF molecule streams, RDF statement streams are fine grain streams

of triples. We envision the possibility to define up to eight different types of RDF statements streams depending upon the kind of information that changes at each stream input, ranging from a completely unspecified to a completely specified RDF triple. In the former case, every new element in the stream is an arbitrary RDF triple; in the latter case, every new element in the stream corresponds to the occurrence of an instance of RDF stream which is totally fixed (e.g., another unidentified vehicle seen at a given sensor). The following table summarizes the eight cases:

Name	Subject	Predicate	Object	Denotation
free	-	-	-	free
bound subject	s	-	-	s
bound predicate	-	p	-	p
bound object	-	-	o	o
free subject	-	p	o	po
free predicate	s	-	o	so
free object	s	p	-	sp
bound	s	p	o	spo

For RDF statement streams it is possible to define continuous queries both in terms of a formal abstract semantics and a concrete query language that implements the abstract semantics (namely *C-SPARQL*), following the path already explored in designing CQL [18] for DSMS.

As for CQL, the abstract semantics of such a C-SPARQL language is based on two data types, RDF statements streams (later on shortly named RDFstream) and **instantaneous RDF graphs** (later on shortly named **tgraph**). The two data types are a direct mapping of stream and relation data types in CQL. C-SPARQL queries are executed as trees of fine-grain operators performing selection and abstraction over streams; their optimization and parallelization can be approached by using techniques which are translated from DSMS systems. In Figure 3 we depict how we expect our C-SPARQL engine to share query plans among different registered queries in order to continuously answer in a throughput-efficient manner.

As for CQL, the abstract semantics of C-SPARQL includes operators of three classes:

- A *tgraph-to-tgraph* operator takes one or more tgraph as input and produces a tgraph as output.
- A *RDFstream-to-tgraph* operator takes a RDF statements stream as input and produces a tgraph as output.
- A *tgraph-to-RDFstream* operator takes a tgraph as input and produces a RDFstream as output.

RDFstream-to-tgraph operators use *sliding windows* [19] over RDF statements streams; their efficient evaluation can use the fact that stream elements enter into windows and then exit from windows sequentially, according to the

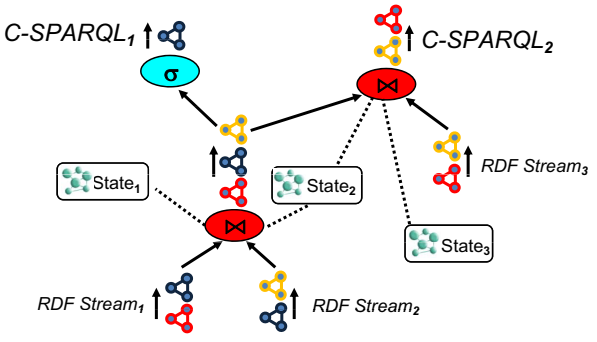

Fig. 3. RDF Statements Stream Reasoning Framework

total order associated with time. RDF data is typically used in the context of ontological languages (e.g., RDF/S and OWL) enabling to describe resources and their properties. The efficient evaluation of multiple queries with several overlapping sliding windows upon both RDF data and language-specific ontological properties is a research challenge currently under investigation; methods presented in [16] can be adapted.

In this framework, scalability will be achieved by distribution and parallelism. Indeed, while each stream should allocated to a given processor, all other operator-based computations can be distributed according to an explicit, well-defined data flow; hence, distributed database methods fully apply.

6 Conclusions and Future Works

In this paper we have presented some preliminary steps toward Stream Reasoning. Our main contribution is an integration architecture, taking advantage of the benefits of both data streams and reasoners, from which two stream reasoning frameworks can be derived.

The one based on RDF molecules is an evolution of the currently available solutions that relies on the possibility to couple DSMSs and state-of-the-art reasoners. This approach requires investigating an appropriate solution for incremental maintenance of time-varying RDF views and engineering throughput-efficient transcoder technology for bridging data streams to RDF Molecules Streams.

The one based on RDF statements is a revolutionary approach to reasoning that requires defining C-SPARQL semantics, studying its computational complexity, defining the concrete C-SPARQL language, and implementing a query processor that heavily exploits the intrinsic characteristic of streams.

Acknowledgements

The work described in this paper is has been partially supported by the European project LarKC (FP7-215535).

References

1. Kiryakov, A.: Measurable targets for scalable reasoning (2007)
2. Garofalakis, M., Gehrke, J., Rastogi, R.: Data Stream Management: Processing High-Speed Data Streams (Data-Centric Systems and Applications). Springer, New York (2007)
3. Kindberg, T., Chalmers, M., Paulos, E.: Guest editors' introduction: Urban computing. IEEE Pervasive Computing 6(3), 18–20 (2007)
4. Arikawa, M., Konomi, S., Ohnishi, K.: Navitime: Supporting pedestrian navigation in the real world. IEEE Pervasive Computing 6(3), 21–29 (2007)
5. Reades, J., Calabrese, F., Sevtsuk, A., Ratti, C.: Cellular census: Explorations in urban data collection. IEEE Pervasive Computing 6(3), 30–38 (2007)
6. Bassoli, A., Brewer, J., Martin, K., Dourish, P., Mainwaring, S.: Underground aesthetics: Rethinking urban computing. IEEE Pervasive Computing 6(3), 39–45 (2007)
7. Fensel, D., van Harmelen, F., Andersson, B., Brennan, P., Cunningham, H., Della Valle, E., Fischer, F., Huang, Z., Kiryakov, A., il Lee, T.K., School, L., Tresp, V., Wesner, S., Witbrock, M., Zhong, N.: Towards larkc: a platform for web-scale reasoning. In: IEEE International Conference on Semantic Computing, ICSC 2008 (August 2008)
8. Fensel, D., van Harmelen, F.: Unifying reasoning and search to web scale. IEEE Internet Computing 11(2), 9695 (2007)
9. Shvaiko, P., Giunchiglia, F., Bundy, A., Besana, P., Sierra, C., Van Harmelen, F., Zaihrayeu, I.: Benchmarking methodology for good enough answers. Technical report, DISI-08-003, Informatica e Telecomunicazioni, University of Trento (2008)
10. Tatbul, N., Çetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load shedding in a data stream manager. In: VLDB 2003: Proceedings of the 29th international conference on Very large data bases, VLDB Endowment, pp. 309–320 (2003)
11. Tatbul, N., Cetintemel, U., Zdonik, S., Cherniak, M., Stonebraker, M.: Exploiting punctuation semantics in continuous data streams. IEEE Trans. on Knowledge and Data Eng. 15(3), 555–568 (2003)
12. Thaper, N., Guha, S., Indyk, P., Koudas, N.: Dynamic multidimensional histograms. In: SIGMOD 2002: Proceedings of the 2002 ACM SIGMOD international conference on Management of data, pp. 428–439. ACM, New York (2002)
13. Chakrabarti, K., Garofalakis, M., Rastogi, R., Shim, K.: Approximate query processing using wavelets. The VLDB Journal 10(2-3), 199–223 (2001)
14. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Commun. ACM 13(7), 422–426 (1970)
15. Ding, L., Finin, T., Peng, Y., da Silva, P.P., McGuinness, D.L.: Tracking RDF Graph Provenance using RDF Molecules. Technical report, UMBC (April 2005)
16. Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. J. Data Semantics 2, 1–34 (2005)
17. Mendelzon, A.O., Rizzolo, F., Vaisman, A.: Indexing temporal xml documents. In: VLDB 2004: Proceedings of the Thirtieth international conference on Very large data bases, VLDB Endowment, pp. 216–227 (2004)
18. Babu, S., Widom, J.: Continuous queries over data streams. SIGMOD Rec. 30(3), 109–120 (2001)
19. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In: PODS 2002: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, pp. 1–16. ACM, New York (2002)