

Tags can be useful in many applications. Tags on streaming data can enrich existing stream-based applications, e.g., [23–25], and can enable and inspire novel useful services as described in Section 1.2. Other ways of leveraging tags associated with streaming data may include: (1) *Stream data tracking*, where tagged objects can be located and tracked unambiguously. (2) *Creation of rich user profiles*, where information about a user’s interests, mood, observations, and character can be revealed based on the tags employed by him or her in real time, and used in privacy preservation or tailored services. (3) *Exploration and browsing of streaming data*, which can be achieved by exploiting tags as a navigation mechanism allowing users to find related streaming data based on the tags (see Section 4.4.1). (4) *Social communication*, where by allowing other people to tag a specific subset of real-time data with their own tags, one can find out what different people think about the same piece of information. For instance, one scientist’s opinion (expressed via a tag) on real-time measurements in an experiment might vary significantly from the tags attached by other researchers. In general, we envision stream tagging being useful in almost any application in which streams are produced or consumed.

1.3.3 Diversity-Aware Query Processing

Most modern query optimizers determine a *single* “best” plan at compile time for executing a given query [26]. The execution cost for alternative plans is estimated and the one with the overall cheapest cost is chosen. The cost typically is estimated based on the *average statistics of the data as a whole*, as the objective is to find *one* plan for all data. Such optimization approach largely relies on the *uniformity* of attribute values. As a result, the single plan approach ignores the fact that various data tuples from the same stream may have distinct statistical properties (e.g., frequencies, correlations, etc.). While in some cases such simple and “monolithic” approach to execution plan selection is adequate, the strong assumption of uniformity is often

adopts a *divide-and-union* approach. A relation is partitioned according to selectivities, and subsequently the query is rewritten as a union of constituent queries over the computed partitions. The approach presented in [29], however, only focuses on the partitioning algorithm, rather than a systematic approach to generate different plans for different subsets of data – the focus of our work. In practice, the lack of a comprehensive system support for concurrent multi-plan execution may reduce (or completely cancel out) the effectiveness of such query processing strategy. The authors in [29] also do not address the issue of overhead associated with partitioning (classification) incurred at runtime when the data arrives and the important issue of concurrent multi-plan runtime execution efficiency. Our work addresses these issues and employs a very efficient novel “Self-Routing Fabric” data structure for efficient runtime execution.

Conditional plans [70] generalize serial plans by allowing different predicate evaluation orders to be used for different tuples based on the values of certain attributes. This class of plans can be beneficial when the attributes are highly correlated, and when there is a large disparity in the *acquisition* and evaluation costs of the predicates. Conditional plans primarily focus on often selecting a *single* and very *cheap* to *acquire* partitioning attribute, since query processing is done in the context of sensor networks. Such attribute is not necessarily the “best” splitting attribute in a more general context of DSMSs. In that respect, query mesh is a more general model that can employ an arbitrary number of attributes for partitioning (classification) of data. *QM*, being a general model, can exploit the data security and the tagging metadata (described in Chapters 3 and 4) to partition data into distinct data subsets.

A common problem with pre-computed solutions is that they might become inefficient over time. One approach to address inaccuracy or potential changes in an environment during query execution is through *eager re-optimization* and *runtime adaptation* of execution, e.g., [34, 71, 72]. Systems like IBM’s LEO (LEarning Optimizer) and more recently Microsoft SQL Server use monitoring and feedback to repair

incorrect cardinality estimations and statistics [73–75]. For a survey of adaptive query processing techniques, we refer the reader to [76].

Some works have proposed robust and uncertainty-aware solutions for query optimization e.g., [77–79], but they primarily focus on a *single*-plan strategy for query execution.

Proactive re-optimization in Rio [78] is a robust query optimization technique based on the concept of bounding boxes. Bounding boxes specify range of values that a parameter can take, thus representing uncertainty. The optimizer finds a (set of) plan(s) that behave well in the bounding box, and at run-time, if observed statistics fall inside the bounding box, the best plan in that box (based on the latest statistics) is chosen, else the re-optimization is invoked.

Error-aware optimization (eao) [80], similar to Rio, makes use of intervals over query cost estimates. Eao, however, focuses on memory usage uncertainty rather than selectivity uncertainty.

Babcock et.al. [77] tackle cardinality estimation uncertainty and consider a probability distribution over possible selectivities instead of a point estimate of selectivity. Using probability distribution, the optimizer selects the appropriate query plan after considering the relative importance of predictability vs. performance preference of the user. Prior to optimization, the user selects the trade-off between the two goals of predictability and performance (which could be at odds sometimes) to find the appropriate query plan.

Chu et al. [81] describe the least expected cost optimization technique. Here, instead of finding the lowest cost plan for the expected values of the parameters, the optimizer attempts to find the plan that has the lowest expected cost over the different values the parameters can take. The goal here is to find a “conservative” plan that is likely to perform reasonably well in many situations, rather than a more “aggressive” plan that may work better if the cost estimate is accurate, but much worse if the estimates are slightly off.

Ioannidis et al. [82] present parametric query optimization method, whereby multiple alternative plans are identified at compile-time, after which an actual *single* plan is chosen at run-time, when the actual parameter values are known.

A large body of work has been dedicated to extend support for uncertain data inside databases [83–86] including several efforts in building systems for managing uncertainty [87–89]. While many of these works study efficient algorithms for query processing on uncertain data, none of them actually consider uncertainty in the query processing itself, which is the focus of our paper.

To the best of our knowledge, none of the existing works tackle the problem of uncertainty when multiple query execution plans are employed concurrently for processing of distinct subsets of data.

2.3 Adaptive Query Processing Techniques

Related to our work are several techniques from adaptive query processing [3,6,76]. Here, at different times, tuples may be processed differently, as data statistics or system environment change. Similar to compile-time optimization, most adaptive query processing works still focus on adapting a single query plan as data properties and system conditions change at runtime [73,78].

A very recent survey of systems and techniques for adaptive query processing is given in [90]. Previous work on adaptive query processing considers primarily traditional relational query processing. One technique, which is being incorporated into commercial databases, is to collect statistics about query subexpressions during execution and to use the accurate statistics to generate better plans for future queries [73,91]. Two other approaches [92,93] re-optimize parts of a query plan following a (blocking) materialization point, based on accurate statistics collected on the materialized subexpression.

Convergent query processing is proposed in [68,94]: a query is processed in stages, each stage leveraging its increased knowledge of input statistics from the previous

stage to improve the query plan. The algorithms proposed in [68,94] do not extend to continuous queries and provide no guarantees on convergence. Reference [95] explores the idea of moving execution to different parts of a query plan adaptively when input relations transferred from remote nodes incur high latency. The POP approach adds checks to conventional query plans in DBMSs to detect sub-optimality during query execution, invoking re-optimization if required [79].

The previously-mentioned Eddies architecture [4, 33, 34, 65, 96, 97] enables fine-grained adaptivity by eliminating query plans, by instead routing each tuple adaptively across operators that need to process it. Eddies [34], which can potentially adapt at the tuple granularity, is observed to mostly be using a single plan for nearly all tuples as also indicated in [28]. Closely related to the *QM* paradigm is the content-based routing (CBR) extension of Eddies [28] that considers not only properties of operators (such as their selectivities and backlog) but also the content of the data. CBR, an extension to Eddies, however inherits several problems associated with Eddies, such as expensive on-the-fly decision-making and often unnecessary tuple-level granularity of adaptivity. [33] adds batching to the Eddies routing to reduce the tuple-level routing overhead. This work differs from ours in that it is still “route-less”. Further, the batching process is still neither content- nor route-based: batches of k tuples (i.e., continuous chunks of tuples that happened to arrive together in time) are routed together to aim to reduce the rather significant overhead associated with Eddies. Our *QM* solution is more coarse-grained than Eddies in that at a given point in time one execution plan is followed by all input tuples for a continuous query.

Apart from Eddies, the CAPE [8] and Gigascope [5] DSMSs support adaptive query processing over data streams. CAPE supports adaptive processing at the level of operators, e.g., within a join operator, as well at the level of query plans, e.g., switching among different plans for a query. CAPE also supports adaptive placement of query plan fragments across machines in a parallel processing environment. The two-level query processor in Gigascope can adapt the partitioning of work between the two levels, based on the characteristics of the input streams.

2.4 Streaming Metadata

Punctuations as sub-stream delimiters inside data streams have been first presented in [98]. *PJoin* [99] and *PWJoin* [100] apply punctuations to achieve join optimizations on streaming data. [101] uses punctuation-like annotations to inject dynamic schema-knowledge into XML stream to facilitate query optimization and out-of-order processing. [102] uses punctuations for execution safety checking of continuous join queries (CJQs). Punctuation uses *continue* to expand beyond their original semantics – delimiting epochs in the stream.

The authors in [103] propose a feedback mechanism based on punctuations that flow against the stream direction, and carry an intent, as opposed to embedded punctuations [104, 105]. DSMSs have focused on collecting and distributing feedback information by statically placing monitors in the query plan and directly sending parameter changes to operators. The approach in [103] differs significantly in at least two aspects: (1) the use of punctuation to convey the feedback messages, and (2) giving operators the ability to create, consume, and propagate feedback, eliminating the need for centralized managers.

AT&T’s Gigascope DSMS includes a type of punctuations – called “heartbeats” – that signal the passing of time, but their work does not exploit punctuations as feedback mechanisms for optimization [106].

This thesis is the first work proposing to use punctuations as (1) security constraints to enact access control policies, (2) as semantic labels (tags) to attach additional semantics to streaming data, and (3) as routing “itineraries” to process various subsets of data using different execution plans.

2.5 Learning Techniques

Related to the *QM* concept presented in this thesis are several techniques in machine learning. There has been plenty of work on building mining models over continuous data streams, including clustering [107, 108], decision trees [109] [110],

nearest neighbors [111], and heavy hitters [112, 113]. New algorithms have also been proposed for maintaining statistics over data streams, e.g., samples [114], histograms [115], and quantiles [116].

A number of algorithms have been proposed in the literature for extracting knowledge from data, using clustering [117, 118], classification [109, 110, 119], frequency counting [120, 121] and time series analysis techniques [122, 123]. These techniques can be integrated into the classifier component of the QM , the subject we plan to investigate further.

Decision tree construction is an important problem in data mining [124–127]. Most of the proposed algorithms address the problem of decision tree construction for static data. The key issue in mining on streaming data is that only one pass is allowed over the entire data. Moreover, there is a real-time constraint, i.e., the processing time is limited by the rate of arrival of instances in the data stream, and the memory available to store any summary information may be bounded. For most data mining problems, a one pass algorithm cannot be very accurate. The existing algorithms typically achieve either a deterministic bound on the accuracy [108], or a probabilistic bound [110]. A good survey of data mining techniques for streaming environments is presented in [128].

Several methods have been proposed to deal with time changing concepts [109, 129, 130]. The two basic methods are based on *temporal windows*, where the window fixes the training set for the learning algorithm and *weighting tuples* that ages the training tuples by shrinking the importance of the oldest tuples. The time window method can be improved by adapting its size [129, 131].

VFDT [110] is a very fast decision tree algorithm for data-streams. The main innovation in VFDT is the use of the Hoeffding bound to decide when a leaf should be expanded to a decision node. Later, VFDT has been extended with the ability to detect changes in the underlying distribution of the examples. CVFDT [109] is an algorithm for mining decision trees from continuous-changing data streams. CVFDT works by keeping its model consistent with a sliding window of the most recent ex-

amples. When a new example arrives it increments the counts corresponding to the new example and decrements the counts to the oldest example in the window which is now forgotten. Each node in the tree maintains sufficient statistics. Periodically, the splitting-test is recomputed. If a new test is chosen, CVFDT starts growing an alternate subtree. The old one is replaced only when the new one becomes more accurate.

Other learning techniques in artificial intelligence, such as neural networks [132], intelligent agents and reasoning [133], intelligent information systems [134], logic and logic programming [135], planning and scheduling [136, 137], bayesian networks [138], genetic programming [139] have also received a lot of attention from the research community in the recent years.

Works dealing with uncertainty in classification and machine learning [140] focus primarily on the prediction accuracy of the models. Classical versions of classification algorithms typically are not designed to handle uncertainty [141]. To overcome this limitation, probabilistic decision trees [142], bayesian decision trees [143], and classifier ensembles [144] have been proposed to deal with classification of data with missing, imprecise, or updated attribute values.

2.6 Security and Access Control Enforcement

Agrawal et al. have coined the concept of Hippocratic databases [145] to incorporate the privacy protection within relational DBMS. Hippocratic databases use privacy metadata to represent the data owner's privacy preferences and the the data collector's privacy policies. The data is returned to users only when the policies meet the preferences. The work focuses on relational databases only and does not address the challenges present in the streaming context.

The problem of access control in dynamic environments has raised significant interests in research community in recent years [146–148]. [149] extends RBAC model to Temporal-RBAC, which supports periodic role enabling and disabling and tempo-

ral dependencies among permissions. GEO-RBAC [150] extends RBAC model with spatial awareness. For most of these access control models, however, the changes in the policies do not get reflected on the results until the query is re-executed after the change. While the focus of our work is to enforce access control throughout the long running time of continuous queries in a DSMS.

The notion of continuous access control has been introduced by Ravi Sandhu et al. as part of the *UCON* model [151, 152]. To the best of our knowledge, apart from the initial theoretical paper, our on continuous access control enforcement in DSMSs is the first real instance of the *UCON* model. One of the reasons for the lack of real systems is that it is difficult to implement in practice [151]. We believe that we are the first to do so.

Fine-grained access control in relational databases has received a lot of attention recently [153–155]. Fine-grained access control allows control of access at the granularity of individual rows, and to specific columns within those rows and is often required in many database applications. Wang et.al. [153] design a labeling scheme to hide information in a database. To answer queries the authors propose a query modification approach to evaluate the queries over tables with masked cells. Chaudhuri et.al. [154] propose a model for fine-grained authorization based on adding predicates to authorization grants. The model supports predicated authorization to specific columns, cell-level authorization with null-ification, authorization for function/procedure execution, and grants with grant option. The model also incorporates other novel features, such as query defined user groups, and authorization groups, which are designed to simplify administration of authorizations. The model is designed to be a strict generalization of the current SQL authorization mechanism. Kabra et.al. [155] make an observation that the majority of models for fine grained access control follow a view replacement strategy which suffers from the overhead of the access control predicates when they are redundant and potentially may leak information through channels such as user-defined functions, and operations that cause exceptions and error messages. The authors propose techniques for redundancy re-

removal and define when a query plan is safe with respect to UDFs and other unsafe functions. To address the potential information leakage, the authors propose techniques to generate safe query plans. To the best of our knowledge, none of works on fine-grained access control address the simultaneous enforcement of multiple policies (server side and client side) and typically consider a static relational database context instead of dynamic data streams – the focus of our work.

Another area of related work is the context-awareness in access control and context-aware adaptation of access-control policies, e.g., for crisis management (or emergency). The main idea here is to employ contextual parameters as inputs to the access control model (e.g., a context-sensitive RBAC model [156]). In our paper, we accommodate the requirements of context-aware access control by providing support for: (1) generation of security punctuations based on the real-time context data streams, and (2) support for the immediate enforcement of security policies to tackle emergency situations.

2.7 Tagging Methods

There are several ongoing projects that deal with annotation propagation and management for scientific databases, e.g., DBNotes [157], Mondrian [158], bdbms [159], and MMS [160]. Social bookmarking systems, such as *Flickr* [161], *Delicious* [162] and *Technorati* [163] support annotations of web resources and images with free-text keywords. For more examples of tagging systems and their taxonomy, we refer the reader to [164]. To the best of our knowledge, none of these existing works address the problem of tagging in the context of dynamic data stream environments.

Chi et al. [165] study the entropy of tagging systems, in an effort to understand how tags grow, and how the groupings of tags change over time and affect browsing of data. Halpin et al.’s work [166] looks at the nature of tag distributions with information theoretic tools. There has been some work on association rules in tagging systems, including [167, 167] and [168]. [168] primarily focuses on prediction of tags.

Oldenburg et al. [169] look at how to integrate tags across tagging systems by using Jaccard measure and discuss different types of tagging systems: social bookmarking, research paper tagging systems, but not DSMSs.

Research on self-describing streaming XML which can be viewed as “data tags” has received a lot of attention in recent years [170–172]. XML processing is typically more expensive compared to traditional stream data processing, and requires a special XML stream management functionality (in addition to the XML-aware optimizer and executor). Our proposed tagging approach is simpler in design and more light-weight compared to streaming XML, while at the same time it provides support for rich user-based tag semantics.

Relational data-bases have had an extraordinarily successful history of commercial success and fertile research. It is not surprising, therefore, that database researchers have attempted to understand annotations and “tagging” in the context of relational databases [157].

One of the biggest challenges in relational databases is the correct propagation of annotations through queries’ pipelines. This is similar to the problem we’ve discussed in the context of tag-aware query processing. In [157], a practical approach is taken to handling annotation in which an extension of SQL is developed supporting explicit user control over the propagation of annotations. The idea is to allow the user to control the flow of annotations by adding propagation instructions to the SQL query language. In our implementation, the tagging system performs (by default) the system-driven propagation, when processing tag-aware continuous queries. Adding support for user preferences regarding tag propagation in tag-aware queries is a subject of our future work.

Most of the work on annotations of relational data focuses on annotating individual values in a table. Geerts et al. [158] have taken a more sophisticated approach and provide support for annotating associations between values in a tuple. For example, in a query one might want to annotate fields A and B in the output with information that they came from input table R , and the fields B and C with information

that they came from table S . The authors introduce the concept of a “block” – a set of fields in a tuple to which one attaches an annotation and a “colour” which is essentially the content or some property of the annotation. They investigate both the theoretical aspects and the overhead needed to implement the system. Our approach supports various tagging granularity by using regular expressions in the *Applicability* field in the *tick-tags*, and to maintain the tags’ “lineage” we employ the streaming *stix* concept.

We are unaware of any work that addresses the problem of real-time data tagging in the context of DSMSs and provides support for both explicit and implicit tag querying. Furthermore, our proposed approach is unique in that it is stream-centric: tags attached to streaming data are interleaved with the actual data tuples in the data streams, and the processing of these streaming tags is encapsulated inside the tag-based query operators that can be combined with regular continuous query operators.

2.8 Summary

In this chapter, we have described work related to the concepts and algorithms proposed in this thesis, including an overview of various DSMSs, static and adaptive query optimization approaches, existing streaming metadata techniques in DSMSs. We have also discussed relevant access control solutions from the security area and the tagging and annotation approaches in various systems.

3 SECURITY AND ACCESS CONTROL FOR STREAMING DATA

In this chapter, we address the problem of *continuous access control enforcement* in dynamic data stream environments, where *both* data and query security restrictions may potentially change in real-time and must be enforced online.

We present the *FENCE* (short for *Continuous Access Control Enforcement in Dynamic Data Stream Environments*) framework that effectively addresses this problem. The distinguishing characteristics of *FENCE* include: (1) the *stream-centric* approach to dynamic security, (2) the *symmetric* security model for both continuous queries and streaming data, and (3) two alternative *security-aware query processing* methods, that can optimize the execution based on data-related as well as security-related selectivities. In *FENCE*, both data and query security restrictions are modeled symmetrically in the form of security metadata, called “security punctuations”. Security punctuations stream together with the data instead of being persistently stored on the server. We distinguish between two types of security punctuations, namely, the *data security punctuations* (or short, *dsp*s) which represent the access control policies of the streaming data, and the *query security punctuations* (or short, *qsps*) which describe the access authorizations of the continuous queries running on the server. With respect to the problem of efficient execution of continuous queries, we propose and compare two security-aware query processing methods, namely: (1) the *Security Filter Approach (SFA)*, and (2) the *Query Rewrite Approach (QRA)*.

The rest of this chapter is organized as follows. In Section 3.1, we motivate the need to address the problem of continuous and online access control enforcement for streaming data. We give the problem definition in Section 3.2. Section 3.3 presents the *FENCE* architecture. Section 3.4 describes our security model with data and query security punctuations and their semantics. Section 3.5 presents the alternative

security-aware query processing methods. We describe the results of our experimental study in Section 3.6. We conclude in Section 3.7.

3.1 Security in Data Stream Management Systems

Due to recent developments in pervasive and ubiquitous computing, many enterprises begin to provide high-quality services based on real-time data, e.g., patient monitoring, location-based services and ubiquitous social networking [2, 173, 174]. The information in such applications arrives in the form of infinite data streams to a DSMS, where continuous queries are evaluated.

One of the biggest challenges in such dynamic data stream environments is the access control enforcement – the ability to permit or deny a request to perform an operation (e.g., a read operation). Given the *long-running* nature of continuous queries, the content of the streaming data and along with it its “sensitivity” may change frequently over the lifetime of query execution. Furthermore, queries on the server may also experience frequent changes in their access control privileges while being executed. Such changes in security privileges may be due to mobility and varying context of the users receiving the results of continuous queries: query results may be accessed via mobile phones, PDAs or iPhones from any place and at any time. Clearly, the users sending their streaming data can be rightly concerned about possible unauthorized accesses to their real-time information and potential violations of their privacy. One of the major challenges here comes from the fact that the security policies of both data and queries can be concurrently very dynamic.

Example 1: *Ubiquitous healthcare system.* Healthcare systems support real-time monitoring and access to vital signs data of patients by doctors, emergency personnel, and pharmacies. Consider a physician executing a continuous query Q that monitors the health state of his patients, e.g., heart rate, blood pressure, etc. Over time, while the query Q is being executed, the physician may continuously acquire different

roles¹, which may have different access privileges, e.g., a hospital employee (R_1), a doctor with unrestricted access (R_2), or a doctor with restricted access (R_3). Possibly, multiple combinations of these roles can be active at any time depending on the policy and the doctor’s context. While working in the emergency room, the doctor’s active set of roles may be: $\{R_1, R_2\}$. When entering an insurance building to settle a claim, the doctor’s active set of roles immediately changes to: $\{R_3\}$. In the evening, when the doctor comes back home, his active role set becomes: $\{R_1\}$. The patients, transmitting their data through their monitoring devices, should have the ability to continuously regulate in which role their doctor can access their real-time health information.

Example 2: Location-Based Services. Recent improvements in location-based technologies and the drop in prices of location-tracking devices have spurred a new wave of mobile services, such as location-based services and geo-social networking applications [19]. Such applications naturally raise privacy concerns. Users consider their physical location and travel patterns highly privacy-sensitive and demand solutions that are able to protect their information. Therefore, it is essential to provide support for users to be able to frequently change their access control policies based on their preferences, to restrict who can “see” their real-time information (e.g., where they are, whom they are with, or what they are doing).

Based on these real-life examples, we can observe that dynamic changes in policies are natural and represent an essential part of an access control environment in data streams. We can also observe that changes in security may arise not only because of (1) the dynamic preferences of the users sending their data (i.e., the data providers) but also from (2) the dynamic privileges of the users receiving the results of continuous queries running on DSMS (i.e., the query specifiers). To the best of our knowledge, our work is the first to address the problem of *online access control enforcement* with *concurrent* dynamic changes in security for *both data and queries*.

¹Here, we assume the system is using a role-based access control (RBAC) model [175].

3.1.1 Challenges

Due to the characteristics of streaming data, there are a number of inherent challenges that make continuous access control enforcement a challenging task.

- *Fast data arrival rate.* A common characteristic of data streams is a high data volume and a rapid arrival rate [2]. It is not feasible to store all data from all streams and take random accesses to the data as it is done in traditional databases. Therefore, the security policies associated with a data must be determined as fast as possible and the speed of the access control enforcement algorithm must be faster than the incoming data rate.
- *Single scan of data.* Due to the massive volumes of data, there may be not enough space to store all streaming data and its security policies (which may be numerous and of fine granularity). Therefore, one scan of data and its security restrictions with compact memory usage is required.
- *Dynamic changes in security.* The widespread usage of portable devices and the users' mobility are likely to lead to frequent changes in transmitted streaming data and possibly its "sensitivity". In addition to that, the mobility and the changing context of the users receiving the results may translate into frequent changes in the access control authorizations. Thus, an access control enforcement mechanism must be adaptive to runtime changes in security.
- *Correctness of enforcement.* The foremost challenge is the prevention of any information leaks that may occur when access is no longer authorized. It is also important to ensure that the access to data is not denied, when an access privilege has, in fact, been granted, especially when it is crucial to see the data *immediately* (e.g., in the case of an emergency). At any time, only the data elements that satisfy both the query and the data security policies at the same time must be returned as query results.

- *Low overhead.* The results in streaming environments are expected to be produced in near-real-time. Since access control enforcement is nothing but an added “overhead” compared to the traditional continuous query processing, its cost must be as low as possible not to decrease the utility of the DSMS.

3.1.2 Our Contributions: The *FENCE* Framework

To address the above-mentioned challenges, we propose the *FENCE* (short for *Continuous Access Control Enforcement in Dynamic Data Stream Environments*) framework that supports the online enforcement of changes in the security policies of the data as well as in authorizations of the continuous queries while they are being executed. *FENCE* employs the *Security Punctuation (SP)* model [38] for both the streaming data and the continuous queries. Furthermore, *FENCE* enables a much richer security semantics for various applications’ needs. These features introduce new technical challenges for which we present our solutions in the rest of this chapter. Our major contributions can be summarized as follows:

- *FENCE* models *both* data-side and query-side dynamic security restrictions *symmetrically* using streaming “security punctuations”² metadata. *FENCE* extends the *SP Framework* [38] scheme by distinguishing between the two types of *sps*, namely, the *data security punctuations (dsps)* and the *query security punctuations (qsps)* to enforce security for both data and queries in a simple and efficient manner.
- *FENCE* framework supports security-aware continuous query processing with combined *dsps* and *qsps*. Compared to [38], which supports only *one* security-aware query processing method, *FENCE* is equipped with two adaptive techniques, namely: (1) the *Security Filter Approach (SFA)*, and (2) the *Query*

²We chose to name the streaming security metadata “security punctuations” (or short *sps*), because by introducing *sps* into data streams, we subdivide (i.e., punctuate) infinite data streams into finite partitions with associated access control policies.

Rewrite Approach (QRA). We discuss the advantages and the limitations of each of the methods, and describe how both methods can support security-aware and compliant query processing, and can adapt to both data as well as security-related selectivities.

- Since in data stream environments, the access control policies may change in the middle of query execution, *FENCE* distinguishes between two types of security policy enforcement semantics, namely the *deferred* and the *immediate* enforcement. In the former, the access control policies are enforced on only the data tuples that arrive *after* the policy change. Alternatively, in immediate enforcement (e.g., in emergency scenarios), the access control is enforced *instantly* including the tuples that have arrived *before* the policy change and are not yet returned as query results. We formally address this issue and provide an efficient solution to support both types of security policy enforcements.
- We have implemented *FENCE* in a general DSMS prototype [8]. Our experimental study shows that *FENCE* efficiently supports access control on data streams with data and query security policy changes and security-related overhead with *sps* is low relative to continuous query execution cost.

3.2 Problem Formulation

To formulate the problem we address in this chapter, we first give the definition for the concept of *continuous query processing* (or CQP for short). In traditional CQP, continuous queries are registered in DSMS, and only the data tuples that satisfy the predicates of the continuous queries are produced as results. We call these predicates – *query predicates* – and formally define CQP as follows:

Definition 3.2.1 (Continuous Query Processing (CQP)) *Suppose that a data element $d=(v_1, v_2, \dots, v_n)$ from a data stream has n attributes and a query predicate $\varphi_Q(attr_1, attr_2, \dots, attr_n)$ on d represents the condition of a given continuous query*

Q . Then, whenever d arrives, the continuous query processing mechanism produces d as a result of Q if and only if $\varphi_Q(v_1, v_2, \dots, v_n)$ is true.

In *Security-Aware Continuous Query Processing* (SA-CQP), in addition to the query predicates, there is an additional type of predicates, called *security predicates*, which determine whether the query may access the arriving data tuples based on the current access control policies. We distinguish between two types of security predicates, namely: (1) the *data-side security predicates*, which represent the data provider's security policies on the streaming data and (2) the *query-side security predicates*, which describe the query specifier's current access authorizations. Continuous queries, registered by a user (i.e., query specifier) implicitly acquire the access authorizations of that query specifier. Consequently, SA-CQP enforces access control on data streams by only producing the results that satisfy both the query predicates and the security predicates at the same time. SA-CQP can be formally defined as follows:

Definition 3.2.2 (Security-Aware Continuous Query Processing (SA-CQP))

Suppose that a data element $d = (v_1, v_2, \dots, v_n)$ from a data stream has n attributes, a query predicate $\varphi_Q(attr_1, attr_2, \dots, attr_n)$ on d represents the condition of a given continuous query Q , and a security predicate $\varphi_S(attr_1, attr_2, \dots, attr_n)$ on d represents a security policy S . Then, whenever d arrives, the security-aware continuous query processing returns d as a result of Q if and only if $\varphi_Q(v_1, v_2, \dots, v_n) \wedge \varphi_S(v_1, v_2, \dots, v_n)$ are both true.

Figure 3.1 visually depicts the SA-CQP concept. Query predicates are denoted by φ_Q , and security predicates φ_S are composed of two types of security predicates, namely the *data-side security predicates* and the *query-side security predicates* denoted by φ_{ds} and φ_{qs} , respectively.

In our work, we address one of the key aspects of security in data stream environments, namely, the *dynamic changes in security policies* (specifically, the changes in access control), while continuous queries are being executed. Dynamic security means that during query execution access control policies affecting the processing

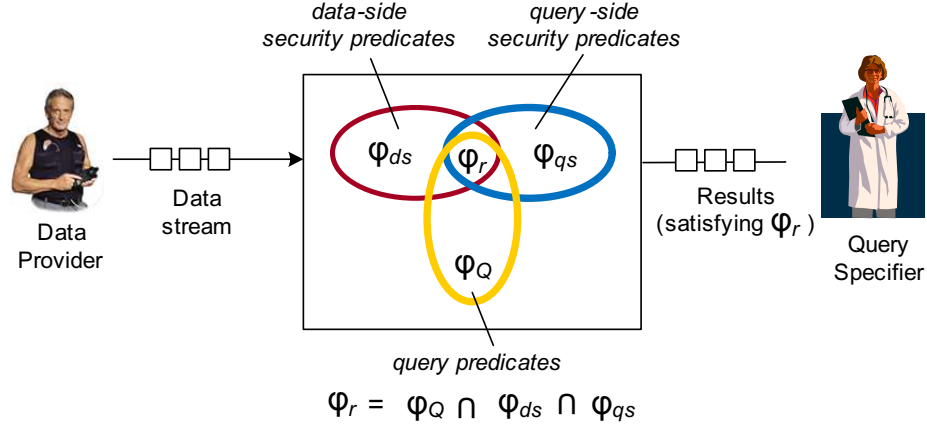


Figure 3.1. Conceptual idea of security-aware continuous query processing (SA-CQP).

(and the results) of the query may frequently change to support the real-time needs of users and the requirements of applications. *Data-side dynamic security* represents the changes in the data providers' security preferences and *query-side dynamic security* represents the changes in the query specifiers' access authorizations. In our work, we provide a solution for SA-CQP in the presence of both types of dynamic security.

3.3 Overview of *FENCE* Framework

In this section, we present the general *FENCE* architecture and then describe a specific instance of the framework that we consider in the rest of the paper.

3.3.1 *FENCE* Architecture

To model dynamic access control in a data stream environment, *FENCE* extends the concept of security punctuations introduced in [38]. *Security punctuations* (or short, *sps*) are meta-data embedded inside data streams that describe the following aspects: (a) who has access rights, (b) to which streaming data objects, and (c) when. Compared to the original *sps* in [38], which only describe the data-side security policies, *FENCE* extends the *sp* paradigm to model both the *data-side* dynamic security policies as well as the *query-side* dynamic access authorizations that may be both

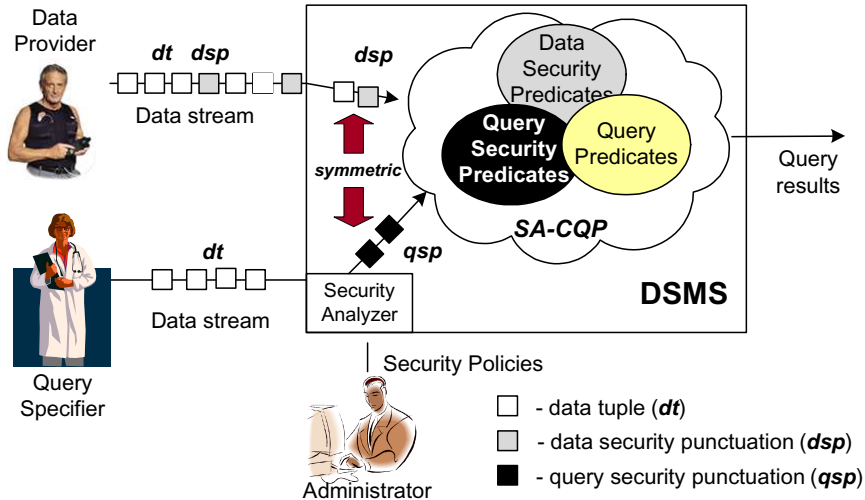


Figure 3.2. Overview of *FENCE* architecture.

continuously changing. By uniformly representing the security settings for data and queries using a single concept, namely the security punctuations, *FENCE* facilitates a simpler security model, code re-use and enables similar security processing for both data and queries. Using streaming *sps*, a DSMS can support *online* flexible, dynamic, and fast access control enforcement over infinite data streams, while queries are being evaluated. We will discuss the concept of security punctuations, as applicable to data and queries, in Section 3.4.

Figure 3.2 shows a high level overview of the *FENCE* architecture. In a typical streaming environment, we distinguish between three types of users: (1) The *data provider* – a user continuously sending his or her streaming data with the interleaved *sps*, that describe the real-time security preferences on his or her streaming data. (2) The *query specifier* – a user who registers a continuous query on the server to be evaluated on the incoming streaming data. As can be seen from Figure 3.2, a query specifier also streams his or her real-time context (via a data stream), based on which *qsps* that describe the real-time access privileges of the continuous query are generated. For example, if a query specifier is a physician and he is out on a lunch break, the current location of the physician (“outside the hospital premises”) will generate a punctuation that limits the data (the health information of his patients)

the doctor can access from his portable device. (3) The *DSMS administrator* is a user responsible for registering security policies that guarantee that correct privileges are given to the queries based on the context of the query specifiers. The *Security Analyzer* component in Figure 3.2 is responsible for generating correct *qsps* according to the organization’s security policy registered by the DSMS administrator. Both data and query-side security are symmetrically modeled by security punctuations. This *symmetry* facilitates a simpler model and similar processing for *both* data and query security metadata inside DSMS. When streaming security punctuations arrive to the system, the query processor interprets *sps* as security predicates by processing the data-side and the query-side *sps* alike, and then, produces results that satisfy both the query predicates as well as the security predicates (as shown in Figure 3.2). In *FENCE*, *dsps* are assumed to be directly generated by the data providers³ and *qsps* can either arrive from the query specifiers (or from a third-party security service) or most likely are generated locally in DSMS by the *Security Analyzer* module based on the query specifiers’ current context⁴. We discuss the different possible scenarios of *sp* generation in Section 3.4.4.

3.3.2 An Instance of the *FENCE* Framework

FENCE is a general framework and is not restricted to any particular data or access control model. But to make our discussion concrete, here, we describe an instance of *FENCE* framework, with a specific data and an access control model that we will consider in the rest of the paper.

³The data security punctuations (*dsps*) may also be generated on the DSMS by evaluating continuous security policy queries on the incoming data streams (see Section 3.4.4), but for simplicity of discussion, we assume that the *dsps* arrive to the DSMS already interleaved with the data.

⁴We assume that the context of the query specifiers is represented by additional incoming data streams, e.g., stream of location updates.

Data and Query Model

We consider a centralized DSMS processing long-running select-project-join (SPJ) queries on a set of infinite data streams. A continuous data stream S is a potentially infinite sequence of tuples that arrive over time. The general schema of tuples in a data stream is described by: $[sid, tid, A, ts]$, where sid is the stream identifier, tid is the tuple identifier, A is a set of attribute values in the tuple, and ts is the timestamp of the tuple. As commonly considered in other streaming systems, e.g., [43, 176], the timestamps of the stream elements are assumed to be ordered. For simplicity of discussion, we consider a single continuous query Q_i , registered by a query specifier in the DSMS, to be executed over data streams A, B, \dots, Z . The security restrictions applicable to the query specifier get implicitly inherited by Q_i . Query Q_i is represented by a query execution plan composed of operators op_1, \dots, op_k , where each operator acquires the security restrictions associated with the query Q_i for which it processes the incoming data tuples.

Access Control Model

An access control policy specifies who has access to which objects and when. In its general form, an access control policy can be described by a triple $\langle object, subject, operation \rangle$. An *object* is an entity that contains the information. Examples of objects in data stream environments are: streams, tuples, tuple attributes and data values. A *subject* may invoke a request to access an object to perform an operation, e.g., a “read operation” on a data tuple. The subjects in *FENCE* are the query specifiers. Subjects acquire the access rights which are the set of privileges that they can hold and execute on an object. In our work, we consider a read right (operation) only. Due to the fact that just about all stream systems are read-only, this is a natural focus. However, the model can be easily extended to support other operations as well, such as update, delete, etc. Access to an object implies the right to use the information

it contains. An access is granted, if the corresponding subject owns a permission for the requested operation. Authorization is the granting of the access permissions.

As an example of an access control model, we consider a *Role-Based Access Control (RBAC)* model [175] in our work, and show how it can be implemented in *FENCE*. RBAC is one of the most well-known and widely-used access control models in modern systems today [175]. The main idea of RBAC is to introduce *roles* as an abstraction layer to decouple subjects and permissions [175]. Under the assumption of using RBAC, the streaming *dsps* describe which roles have currently the access rights to which streaming objects, and the streaming *qsps* depict the current roles of a continuous query. Query specifiers activate their default roles when they sign into the DSMS. We require that each query specifier belongs to at least one role. However, this assignment may change while the query specifier is receiving the results of his or her continuous query.

3.4 Dynamic Security Policy Model

In this section, we present the schema and the semantics of the security punctuations in *FENCE*. We provide examples of various *sps* and describe the scenarios of *sp* generation.

3.4.1 General Security Punctuation Schema

In *FENCE*, we employ security punctuations to model symmetrically both the data and the query-side dynamic security restrictions. Such symmetric model makes SA-CQP simpler and allows security-related code re-use in DSMS. We call the *sps* representing data provider’s preferences for security – the *data security punctuations (dsps)* and the *sps* representing the query specifiers’ access privileges – the *query security punctuations (qsps)*. Figure 3.3 shows a general *sp* schema applicable to both *dsps* and *qsps*. We discuss each field in the *sp* schema next.

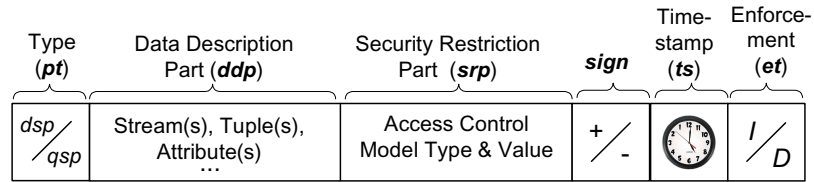


Figure 3.3. General security punctuation schema.

- *Punctuation Type (pt)*: describes whether the punctuation is a data or a query security punctuation.
- *Data Description Part (ddp)*: specifies which object(s) the access control policy applies to, e.g., which stream(s), tuple(s), or tuple attribute(s) [38]. For compactness of storage, we use *regular expressions* to describe objects and their policies inside *sps*.
- *Security Restriction Part (srp)*: denotes both the access control model type and the subjects authorized by the policy. Since we use RBAC in this work (see Section 3.3.2), the *srp* specifies RBAC as the model type and a set of role(s) that are authorized by the *sp*.
- *Sign*: indicates whether the authorization represented by the *sp* is positive or negative (see [177] for more details).
- *Timestamp (ts)*: records the time when the *sp* was generated.
- *Enforcement (et)*: indicates the security policy enforcement setting. We distinguish between two types of enforcement, namely the *Deferred (D)* enforcement and the *Immediate (I)* enforcement. We describe the details of this attribute in Section 3.4.2.

3.4.2 Semantics of Security Punctuations

A security policy may be expressed by one or more *sps* and may apply to zero or more tuples. A set of consecutive *dsps* or *qsps* form a “*batch*” of *sps* which is interpreted as a single access control policy or a complex authorization. All *sps* of

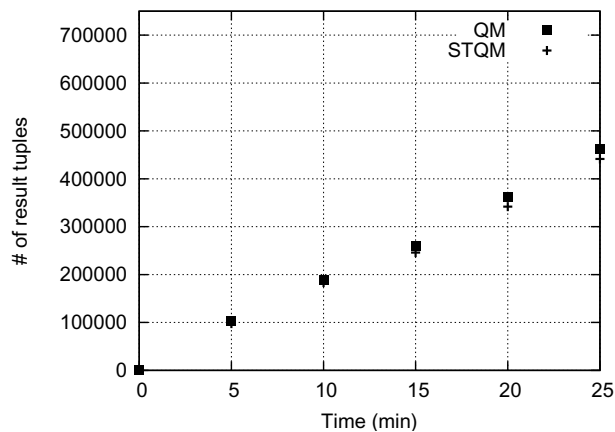


Figure 5.29. Overhead when no adaptation is needed.

no benefit is possible, *ST-QM* is on average between 2.2 - 4.8% worse than static *QM* in the total number of results produced. This result confirms that *ST-QM* approach has detected that changes were insignificant, based on its monitoring and concept drift detection and did not invoke the optimizer. By discarding such insignificant adaptivity cases early, it minimized its adaptivity overhead. This overhead can be further reduced in the system by minimizing the monitoring frequency of both data and execution statistics.

5.11.3 Summary of ST-QM Experimental Conclusions

The main points of our experimental study can be summarized as follows:

1. *ST-QM* can give up to 44% improvement in execution time and output rate.
2. *ST-QM* is highly adaptive to virtual, real and hybrid concept drifts and can result in some cases in up-to 41% improvement compared to non-adaptive *QM*.
3. The runtime overhead of *ST-QM* relative to query execution is small (at most 7%) . The actuation cost of physical adaptivity is nearly negligible resulting in 0.02% of total execution cost.

4. Even if no adaptivity is needed, *ST-QM*'s performance in the worst case will be at most 2-4% slower than of static *QM*.

5.12 ST-QM Conclusion

Here we addressed the problem of *adaptivity* in the multi-plan-based query processing engines. We have presented a *Self-Tuning Query Mesh (ST-QM)* architecture that uses multiple plans for processing different subsets of data, and yet is as adaptive as the “plan-less” systems. *ST-QM* increases the efficiency of query processing in highly dynamic environments, by adapting the multi-plan solution, so that different subsets of data may benefit from different execution plans over time. *ST-QM* approach is unique in that it abstracts the problem of adaptive query processing as a concept drift problem. Such abstraction allows *ST-QM* to discard adaptivity candidates early in the process, if the changes are insignificant to adapt to and thus minimize the adaptivity overhead. The key characteristic of the *ST-QM* approach is that all logical changes to the current *QM* solution get translated into a simple physical operation, namely the classifier change. Our most important contribution is that we have shown in our prototype implementation that *ST-QM* approach can be simultaneously inexpensive and adaptive. Our experimental study indicates that *ST-QM* can adapt to different types of concept drifts very efficiently. Furthermore, the run-time overhead of *ST-QM* execution is fully amortized by the performance benefits of the better multi-plan-based query processing.

Here, we address the problem of adaptive query processing on non-uniform data streams. We propose a *Self-Tuning Query Mesh* infrastructure (or short *ST-QM*) that continuously adapts to data streams' characteristics and to system conditions, e.g., memory, CPU resources availability. The fundamental challenge for self-tuning query mesh is the problem of determining the discrepancy between the previously learned query mesh model and the current model based on the characteristics of the new data and the system condition, what we denote as *optimization concept drift* problem. The self-tuning query mesh has the ability to judiciously determine *when*

and *how* to adapt its infrastructure to accurately match the changed concept of the data streams and employ the best query mesh for the current system conditions. *ST-QM* used a *three-fold* adaptation process - classifier tuning, multi-route configuration tuning, and runtime route tuning - to ensure efficient processing of continuous queries on non-uniform data streams. We have described the tuning techniques in *ST-QM* and have presented detailed experimental evaluation.

5.13 Uncertainty-Aware Query Mesh (UA-QM)

Recent years have witnessed the emergence of novel applications where incoming data arrives in the form of continuous data streams, for example, location-based services, sensor networks and financial tickers. Data in such applications tends to be non-uniformly distributed, and query processing can often benefit from employing *multiple execution plans*, each optimally serving a subset of data with distinct statistical properties. Recently proposed *Query Mesh (QM)* framework implements such *multi-plan* (or *multi-route*²¹) execution paradigm very efficiently. However, similar to most query processing systems, *QM* optimizer assumes all knowledge to be *certain* and *complete* when determining a low cost multi-route solution. Such assumption is unrealistic for streaming environments which are riddled with uncertainty, due to measurement inaccuracies, incomplete or unknown information or data arrival latencies. Here, we focus on the problem of uncertainty in the multi-route query processing context, and propose a novel *Uncertainty-Aware Query Mesh* solution (or short *UA-QM*) to address this problem. The goal of *UA-QM* is two-fold: (1) to model and measure various types of uncertainty to represent real-life scenarios in streaming environments more accurately and (2) to process data in an uncertainty-aware and multi-route fashion. We have implemented our approach in a prototype DSMS, and our experimental evaluation shows the benefits of our proposed *UA-QM* approach.

²¹We use terms “plans” and “routes” interchangeably in our work. Both mean the same thing in the context of this paper.

5.13.1 Problems with Uncertainty Ignorance

Compared to traditional relational databases where the entire dataset is present and so are the complete statistics about it, this luxury is not available in DSMSs. Here, the knowledge about the environment such as data input rate, operator selectivities, attribute values and their distributions is typically incomplete and is continuously changing. Thus, uncertainty naturally arises during query optimization in the streaming context. Most query processors in Data Stream Management Systems (similar to relational counterparts), however, consider all knowledge to be certain and complete during optimization phase, which may significantly limit query performance at runtime [77]. Existing solutions dealing with uncertainty e.g., [77–79] are primarily *single-plan*-based and focus on cardinality estimations for the data *as a whole*. A more complex structure and a different execution paradigm (that employs unique plans for distinct subsets of data) makes these uncertainty solutions insufficient for a multi-route solution like query mesh (see Section 5.14.2 for more detailed explanation).

As a motivating example, consider a geo-social networking application, such as *BrightKite* [18]. Here, an input data stream *people* may be transmitting real-time location updates from the users looking to get together to socialize in a given geographic area. A continuous query Q (shown at the top left in Figure 5.30) is executing to match people based on similar age, interests and location. For simplicity of discussion, we assume that the stream *people* has two distinct data subsets, denoted as the “*city*” and the “*suburbs*” subsets²². Figure 5.30 (the table at the bottom left) shows the selectivities of operators OP_1 - OP_3 for each of the subsets, and the overall selectivity. Here, the selectivities are represented as “certain” point estimates – a common approach in most database systems. Assuming that operators have the same execution costs and only overall selectivities are considered, the best ordering for *people* stream tuples for query Q is OP_2 , OP_3 , OP_1 . However, if we distinguish

²²This could be people living in a city and in the suburbs.

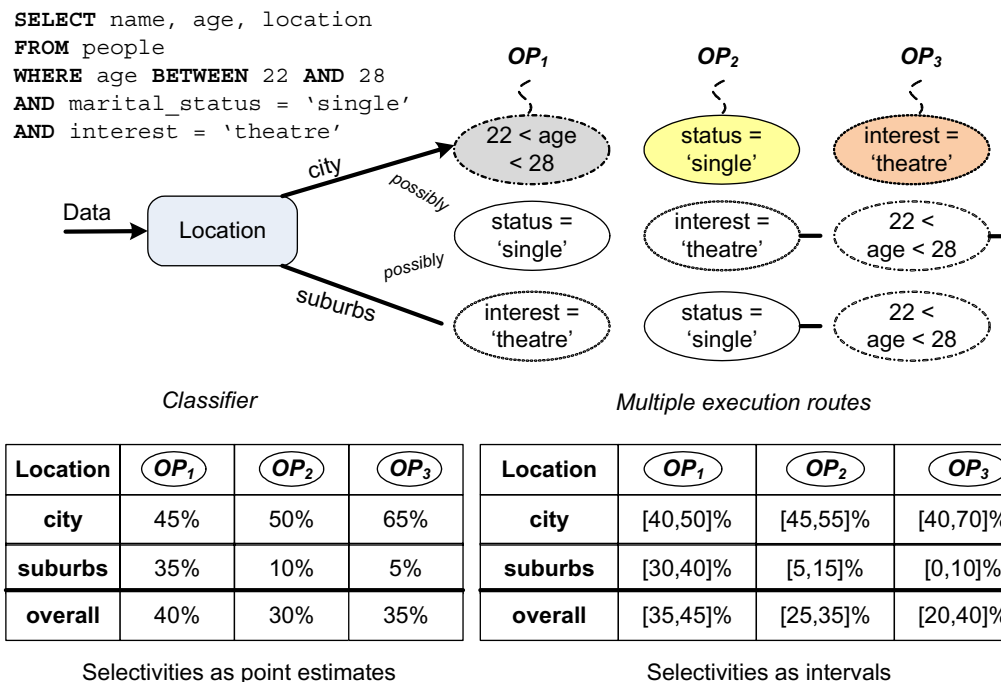


Figure 5.30. Geo-social networking query example.

operators' selectivities based on the different subsets, we can see that for the “*city*” tuples, the ordering OP_1, OP_2, OP_3 will outperform OP_2, OP_3, OP_1 , while OP_3, OP_2, OP_1 will outperform OP_2, OP_3, OP_1 for the “*suburbs*” tuples.

While the above example clearly motivates the benefit of processing different data using different plans, it has a major limitation: it completely ignores uncertainty. For example, uncertainty may be present in the operator selectivities due to varying estimates based on (several) data samples, each roughly approximating the real data, or due to data arrival latency. Uncertainty in operator statistics translates into uncertainty in execution routes. Furthermore, the classifier, which is used to assign routes to data tuples, due to the training set quality or its limited size, may contain some uncertainty as well.

Consider the same example in Figure 5.30 at the bottom right, where instead of certain point estimates, we use *selectivity intervals* to represent uncertainty in

operator selectivities. With uncertain selectivities, it becomes much more challenging to determine which data tuples should be processed using which of the present routes. For example, the “*city*” tuples may benefit from the ordering OP_1, OP_2, OP_3 , but also in some cases, it could benefit from the alternative ordering OP_2, OP_3, OP_1 (depicted by a dashed line) due to the overlap in the operators’ selectivity intervals. Similarly, some “*suburbs*” tuples may benefit from the alternative routes if uncertainty is considered during query optimization.

In addition to uncertainty in routes, classifier may also face uncertainty. For simplicity of presentation, the classifier in our example consists of a single test on the *location* attribute. However, in real-life scenarios, a classifier is likely to have multiple test nodes based on which tuples are assigned to their best routes. Thus, given uncertain routes and uncertain route assignments, as well as the training set only roughly approximating the real data, a classifier is likely to contain some uncertainty as well.

In summary, the problem with many current optimization techniques is that they are: (1) mostly uncertainty-oblivious; and (2) those that are uncertainty-aware, primarily focus on uncertainty in a single query plan execution strategy. Here, we propose to address the open problem of uncertainty in multi-route query processing in the context of data streams.

5.13.2 Challenges

A number of characteristics inherent in streaming environments make the problem of handling uncertainty in a multi-route solution a challenging task.

- *Fast data arrival rate.* A common characteristic of data streams is a high data volume and a rapid arrival rate. Therefore, uncertainty estimation algorithm needs to be as fast as possible and the speed of decision-making regarding uncertainty must be faster than the data incoming rate.

- *Different types of uncertainty.* Uncertainty in a multi-route solution may occur in both routes as well as in classifier which is responsible for assigning tuples to routes. Moreover, uncertainty may be “*absolute*” (with regard to actual values, such as statistics measurements, e.g., order of operators), or it may be “*relative*” (with regard to the best choice among multiple possible alternatives). Thus, an uncertainty mechanism must be able to model and measure various types of uncertainty in both routes and classifier as well.
- *User preferences.* Users executing continuous queries may have different preferences regarding the best way of dealing with uncertainty. Users may want to decide what should be a reasonable tradeoff between certainty vs. possibility (in other words, expectation and ambiguity) when their query is being executed. Thus, uncertainty mechanism should provide support for user preferences with regard to how to handle uncertainty during query processing.
- *Low overhead.* The results in streaming environments are expected to be produced in near-real time. Since the added uncertainty-awareness functionality adds processing and storage overheads (compared to regular “uncertainty-oblivious” query processing), the overhead must be as low as possible not to seriously impact the performance of DSMS.

We address the above-mentioned challenges in the context of data streams and present a solution, based on the multi-route query mesh model [219], which we call *Uncertainty-Aware Query Mesh (UA-QM)*.

5.13.3 Our Proposed Solution: UA-QM

UA-QM contributions can be summarized as follows:

1. *Model.* We propose uncertainty model where both *absolute* and *relative* uncertainties are modeled *symmetrically* for both execution routes and classifier in

a query mesh. Absolute uncertainties are represented using *uncertainty intervals* and relative uncertainties using *belief functions*. The *symmetric* property provides a simpler model and similar uncertainty processing in different components of query mesh. (Section 5.14).

2. *Optimization*. We describe uncertainty-aware optimization algorithms including the computation of multiple execution routes and classifier induction under various uncertainty scenarios (Section 5.15).
3. *Execution*. We discuss how uncertainty is handled at runtime when executing a continuous query in the *UA-QM* framework (Section 5.16).
4. *Experiments*. We have implemented *UA-QM* in a prototype DSMS called *CAPE* [8]. We present our experimental analysis showing the benefits of our proposed approach.

5.14 UA-QM Framework

5.14.1 UA-QM Architecture

Figure 5.31 gives an overview of *UA-QM* architecture which builds on top of the core query mesh framework [27, 219]. We have designed *UA-QM* to be highly modular, enabling uncertainty-awareness functionality to be turned on/off with complete transparency to the core *QM* framework (bottom of the Figure 5.31). The architecture is easily extensible: new algorithms, heuristics and metrics can be added without much disturbance to the rest of the system.

The key components of *UA-QM* include: (1) *QM optimizer* with uncertainty extensions, (shaded half-way in Figure 5.31) and described in Section 5.15, (2) *Belief Space* nodes, used to represent uncertainty in routes and classifier (Section 5.15), (3) *Belief Handler*, a component responsible for resolving uncertainty by taking into consideration user preferences (described in Section 5.16), and (4) *Uncertainty encoding*

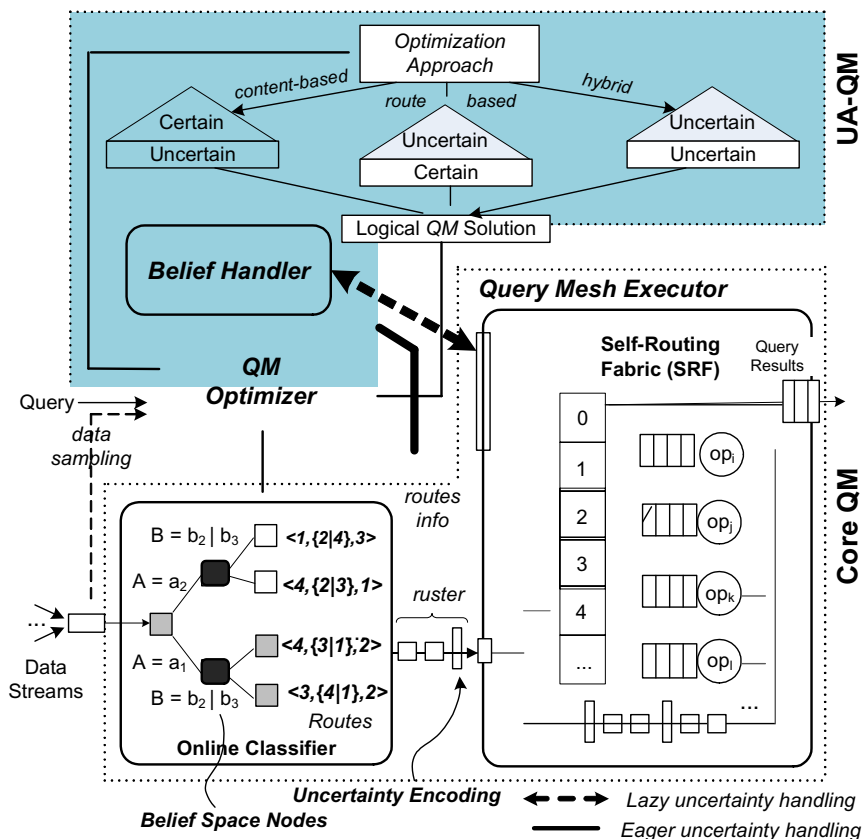


Figure 5.31. *UA-QM* architecture.

used in routes' specifications (discussed in Section 5.16). We present each of these components, their functionality and execution in detail in the rest of the paper.

5.14.2 Uncertainty Cases in Query Mesh

Compared to a single plan solution, where uncertainty may be present in only one route, a multi-route solution may have uncertainty in several routes, as well as in the classifier. For classifier, we employ a *decision tree* model, as it is one of the most commonly used and efficient classification models. Under route uncertainty, we focus on uncertainty in *operators' selectivities*, and under classifier uncertainty, we consider

uncertainty in the *impurity measures* (specifically, in *information gain*²³ [141]) that have a direct impact on the structure and size of the classifier. Next, we describe the three possible uncertainty cases that may occur in a multi-route *QM* solution.

Case 1: *Certain Classifier and Uncertain Routes.* By uncertainty in routes we mean uncertainty in the routes' costs as a result of imprecision in individual operators' selectivities. This leads to ambiguity about the best operator ordering and it may occur when the query mesh optimizer uses a strategy similar to *Content-Learns* algorithm [28], or *Content-Based Approach (CBA)* [219] to find a low cost *QM* solution. The main idea of these optimization strategies is to partition the training dataset into groups based on the similarity of data values first, and then compute the routes for each content group. Different data values may imply unique distributions and statistics, and thus possibly various execution routes. Here, the statistics of the routes may be imprecise, yet the classification based on the training data and the partitions they belong to (determined based on the training data values) is considered to be certain (Figure 5.32(a)).

Case 2: *Uncertain Classifier and Certain Routes.* This scenario is the reverse of the above case. Here, routes are assumed to be certain and the classification may be uncertain (Figure 5.32(b)). This case may occur when the *QM* optimizer uses a *Route-Based Approach (RBA)* [219] to find the best multi-route query mesh solution for a given query. The main idea of this optimization strategy is to compute routes first, using all available statistics (possibly coming from multiple samples of data), and then assign the training data to the existing routes. The statistics used for routes' computations are assumed to be complete and reliable (after being collected over many runs of the same query), yet the training dataset (of limited size) used for the classifier induction may depict real data with some inaccuracy, thus resulting in classification model with uncertainty. This case may also occur when using a large training dataset for classifier induction is prohibitive [141], and some training data

²³Other measures of impurity, such as *entropy* or *gini index* [141] could be used here as well.

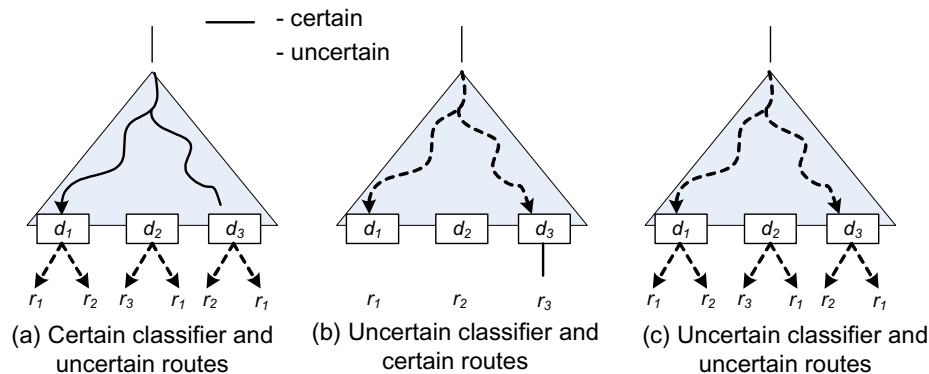


Figure 5.32. Uncertainty scenarios in QM .

must be eliminated from being used in classifier induction – the phenomenon known in machine learning as “pruning” [229].

Case 3: Uncertain Classifier and Uncertain Routes. The third case is the composite of the above two cases, where uncertainty is present in both the routes and the classifier (Figure 5.32(c)). Here, a data tuple may belong to more than one data subset (or a distinct group), and more than one route may be considered to be possible for processing of that subset. This case may occur when a *hybrid optimization approach* is used by the QM optimizer: two QM solutions are computed, one using the *CBA* method and another using the *RBA* method (as described above), and then “merged” to produce the “best” overall QM solution.

5.14.3 Reasoning About Uncertainty

In order to address the problem of uncertainty, we need a method for representing and measuring it. In robust query processing, the common approaches for modeling uncertainty include probability distributions (or short PDs) [77] or bounding boxes (or short BBs) (also known as bounding intervals) [78]. Both methods have their advantages and limitations. PDs, for instance, give an intuitive representation for users to decide how much uncertainty they are willing to “tolerate” when planning an

execution strategy at compile-time. BBs, on the other hand, easily capture variations and imprecisions in statistics, e.g., possible *min*, *max* and expected bounds. It also allows a query processor to check the latest statistics at runtime, and determine which *concrete* execution solution applicable within the bounding interval should actually be employed.

Our uncertainty model includes the strengths of both of the above approaches and enables both user-driven [77] and system-driven [78] responses to handling uncertainty. What sets our model apart from the existing techniques is that we model two types of uncertainty, namely the *absolute uncertainty* and the *relative uncertainty*, and we model them *symmetrically* for both routes and classifier in query mesh. We believe that such two-way uncertainty modeling can represent real-life scenarios more accurately, and the *symmetric* property facilitates a simpler model and similar processing for different components in query mesh.

Informally, absolute uncertainty represents the uncertainty in actual values (e.g., in operator statistics or in attribute impurity estimates), whereas relative uncertainty models the ambiguity about the choice among possible alternatives (e.g., the best order of operators in routes, or the choice of the best splitting attribute in classifier when multiple options are possible). In *UA-QM*, we employ *uncertainty intervals* for modeling absolute uncertainties (see Section 5.14.4) and *belief functions* from Belief Function theory to represent relative uncertainties (see Section 5.14.5).

5.14.4 Absolute Uncertainty

We distinguish between two types of uncertainty intervals in *UA-QM*, namely the *selectivity intervals* and the *classification intervals* to model absolute uncertainty.

Selectivity intervals (or short *SINs*) represent uncertainty in operators' statistics. Table 5.5 shows an example. Here, the selectivity of OP_1 for the distinct subset d_1 might be not known with certainty, but it is known to be between 40% and 50%, and is represented by the interval $[40,50]$. Hence, the interval describes the possibility

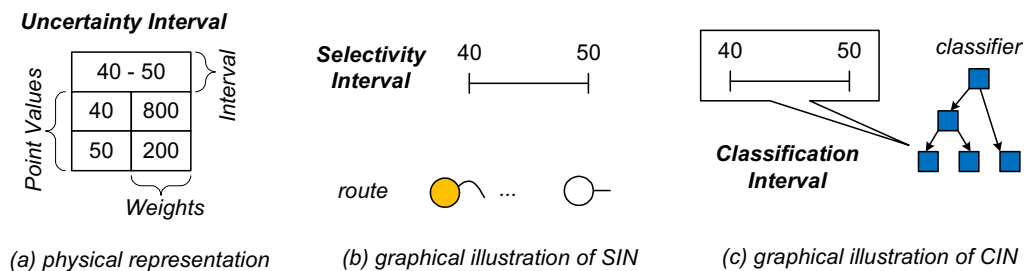


Figure 5.33. Symmetric modeling of *SINs* and *CINs*.

distribution of OP_1 selectivity for subset d_1 . Viewed in this perspective, the entries in Table 5.5 in the column *SIN* are the possibility distributions of the values of selectivities for different subsets of data.

Table 5.5
Selectivity intervals for various subsets.

Stream	Operator	Subset	<i>SIN</i>
S_1	OP_1	d_1	[40,50]
		d_2	[30,40]
		d_3	[35,45]

*Classification intervals*²⁴ (or short *CINs*) represent uncertainty in the *information gain* values of different attributes that are used in determining the best splitting attribute when constructing *QM* classifier. One of the basic steps in decision tree classification is to select the splits based on attributes and data values that are used to predict membership in the terminal nodes of the decision tree classifier (in the context of our work, terminal nodes represent the various execution routes). In general terms, the split at each node found will generate the greatest improvement in predictive accuracy. This is usually measured with an impurity measure, which provides an indication of the relative “homogeneity” of tuples in the terminal nodes of

²⁴A more precise term would be “*impurity measure intervals*”, but we decided to choose a more general name – “*classification intervals*” – to characterize where in *QM* the uncertainty takes place.

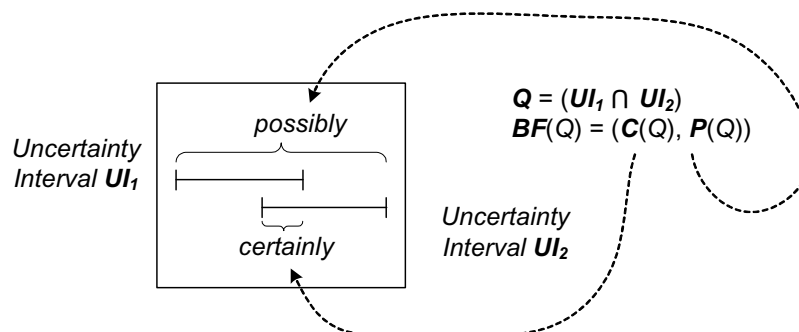


Figure 5.34. Belief function for a relative uncertainty.

the classifier. For example, if all training tuples in each terminal node have identical values, then node impurity is minimal, homogeneity is maximal, and prediction is perfect (at least for the case of training tuples used in the induction of classifier). We omit the discussion of information gain and how it is computed and refer the reader to [141] or any machine learning textbook.

A classifier interval $CIN=[0.4,0.5]$ represents a range of impurity measure values between 0.4 and 0.5 (or in absolute terms $[40,50]$, meaning between 40% and 50%). Due to symmetric property of our model, both $SINs$ and $CINs$ are represented by the same physical data structure (shown on the left in Figure 5.33). Consider, for example, an uncertainty interval $UI [40,50]$ (which could be either a SIN or a CIN). $UI.Interval$ stores the interval value, between 40 and 50. $UI.PointValues$ represent the original point value estimates measured and used in determining the overall interval. $UI.Weights$ correspond to the weights of their respective $UI.PointValues$. For example, in Figure 5.33 in the case of a SIN , the weight of 800 represents the count of tuples that reported to have the point estimate of selectivity equal 40% and the weight of 200 indicates, that 200 tuples contributed to the selectivity point estimate 50%. The weight values could be based on the absolute count of tuples from the samples that have resulted in that particular point estimate value, when the selectivity interval was computed. Alternatively, weights can be represented using relative percent values instead of absolute counts.

5.14.5 Relative Uncertainty

To model *relative uncertainty*, we use the concepts from the *Belief Function Theory* also known as *Dempster-Shafer Theory* [230,231]. Relative uncertainties in routes and in the classifier are expressed in the form of *belief functions*. Belief functions provide a very intuitive way to model ambiguity (compared to classical probability framework), and allow incorporating subjectiveness in uncertainty [77]. Furthermore, if evidences come from multiple sources (e.g., from different samples of data collected at different times), the model provides a flexible and adaptive way to combine those evidences²⁵. Another attractive aspect of belief functions framework is its flexibility – it can be reduced to the Bayesian framework under certain conditions. Figure 5.34 shows a conceptual idea of a belief function. Here, two uncertainty intervals (which could be either *SINs* or *CINs*) are depicted as UI_1 and UI_2 . Under relative uncertainty, the question Q we are interested in is – *how large is the overlap between the two intervals UI_1 and UI_2 ?* This question Q denotes the intersection of the uncertainty intervals and precisely characterizes the relative uncertainty regarding UI_1 and UI_2 . Since both UI_1 and UI_2 are uncertain, the answer to this question can be constructed to have two parts, one relating to the *certainty* C in the answer and the other to its *possibility* P , and symbolically can be expressed as $BF(Q)=(C(Q),P(Q))$. The certainty parameter can be viewed as the expectation and the possibility parameter as the ambiguity. Similar to absolute uncertainties, relative uncertainties, are modelled symmetrically for both routes and classifier in QM as described below.

A *Selectivity Belief Function (SBF)* represents a belief in the intersection of any two *SINs*. A route (query plan) optimization algorithm²⁶ must determine the best order of operators. If any of the operators' *SINs* are overlapping, this translates into the uncertainty about which operator should come first. Figure 5.35 illustrates the possible cases for uncertainty intervals' intersections²⁷: *UIs* may be completely *non-*

²⁵Here, we use a basic form of belief functions. More advanced features like adding subjectiveness to evidence, etc., we reserve for our future work.

²⁶This can be any of the state-of-the-art techniques from the literature, e.g., [58, 66, 209, 210].

²⁷Due to model symmetry, the same logic applies to *CINs* as well.

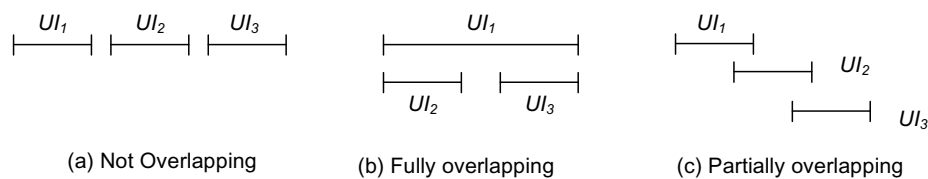


Figure 5.35. Cases for uncertainty intervals' overlaps.

overlapping (5.35a), *completely-overlapping* (5.35b) or *partially overlapping* (5.35c).

For overlapping selectivity intervals, *SBFs* are computed for the following three cases:

- (1) SBF_1 : operator with SIN_1 must come first
- (2) SBF_2 : operator with SIN_2 must come first
- (3) SBF_3 : either of operators may come first

Figure 5.36 shows a concrete example of partially overlapping *SINs*. The values of *SBFs* for this example are as follows: $SBF_1 = (800,1000)$, $SBF_2 = (400,1000)$, and $SBF_3 = (800,2000)$ when expressed in absolute terms, and after normalization: $SBF_1 = (0.8,1)$, $SBF_2 = (0.4,1)$, and $SBF_3 = (0.8,2)$.

Classification Belief Functions (CBFs) represent beliefs in the relative intersections of *CINs* and are modeled similar to *SBFs*. Thus, we omit the details of *CBFs* computation, as it is the same as for *SBFs*. *CBFs* represent relative uncertainty about impurity measures for various attributes, which translates into uncertainty about the best order of splitting attributes when computing the classifier.

In *UA-QM*, Belief functions (both *SBFs* and *CBFs*) are resolved to concrete answers, e.g., specific classification nodes and operators in the routes based on user preferences with respect to uncertainty. Belief functions can be resolved in an “eager” and a “lazy” manner during execution as described in Section 5.16.

5.15 UA-QM Optimization

In this section, we describe uncertainty-aware multi-route query optimization algorithm. We consider three possible uncertainty cases presented in Section 5.14.2.

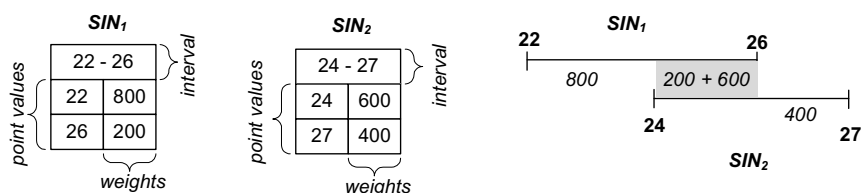


Figure 5.36. Example of computing SBF.

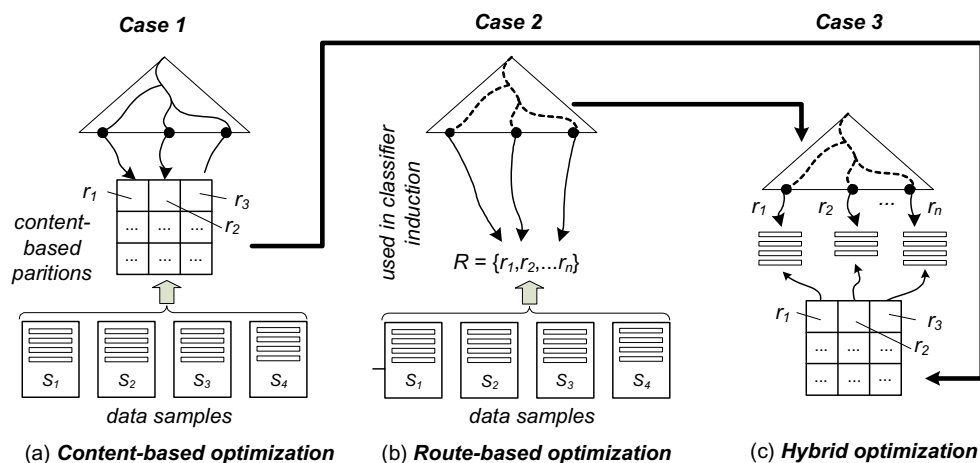


Figure 5.37. UA-QM optimization approaches.

Figure 5.38 illustrates the pseudocode for the overall uncertainty-aware multi-route query optimization algorithm and Figure 5.37 visually depicts the cases.

Case 1: Certain Classifier and Uncertain Routes: As we have previously stated, this case may occur when *QM* optimizer uses a *Content-Based Approach* (CBA) to find a good query mesh (Figure 5.37(a)). Using CBA, the algorithm first divides data into partitions based on similarity of values (Figure 5.38, Line 2), and then computes the execution routes for each content-based partition (Figure 5.38, Line 3).

Computation of Uncertain Routes: The pseudocode for the procedure computing uncertain routes is shown in Figure 5.39. It starts off by computing selectivity intervals for each operator and each content-based partition (Line 1). The operators are then ordered by the monotonically increasing selectivity intervals (Line 2). After the *SIN*s

```

UA-QM-Optimization ( $T$  training dataset represented
by a collection of data samples  $\{k_1, k_2 \dots k_n\}=T$ )
// Case 1: Certain Classifier and Uncertain Routes
01 if (optimization method == Content-Based)
02    $D = \{d_1, d_2 \dots d_n\}$  // content-based partitions
03    $R_u = \text{ComputeUncertainRoutes}(T, D)$ 
04    $C_c = \text{InduceCertainClassifier}(T, D)$ 
05   return new UA-QM( $C_c, R_u$ )
// Case 2: Uncertain Classifier and Certain Routes
06 else if (optimization method == Route-Based)
07    $(R_c, D) = \text{ComputeCertainRoutes}(T)$ 
08    $C_u = \text{InduceUncertainClassifier}(T, D, R_c)$ 
09   return new UA-QM( $C_u, R_c$ )
// Case 3: Uncertain Classifier and Uncertain Routes
10 else if (optimization method == Hybrid)
11   Let  $UA-QM_1$  = solution from steps 1-5
12   Let  $UA-QM_2$  = solution from steps 6-9
13   return new MergeUAQMSolutions( $UA-QM_1, UA-QM_2$ )

```

Figure 5.38. UA-QM Optimization.

```

ComputeUncertainRoutes ( $T$  training dataset,
 $D$  content-based partitions)
01  $SIN = \text{ComputeSelectivityIntervals}(T, D)$ 
02  $oSIN = \text{OrderSelectivityIntervals}(SIN)$ 
03  $uSIN = \text{GetOverlappingSelectivityIntervals}(oSIN)$ 
04  $SBF = \text{ComputeSBFsForSelectivityIntervals}(uSIN)$ 
05  $uR = \text{OrderOperatorsWithSINsAndSBFs}(uSIN, SBF)$ 
06 return  $uR$ 

```

Figure 5.39. Computing uncertain routes.

```

ComputeSelectivityIntervals ( $T$  training dataset,
 $D$  content-based partitions)
 $SIN$  -- a hashtable storing  $SIN$ s of all
operators for different subsets of data
01 for (each operator  $OP$ )
02    $SIN[OP] = NULL$  // No precomputed statistics
      // Compute selectivities using different samples
03   for each sample  $k_i \in T$ 
04     for each partition  $d_j \in D$ 
05       compute selectivity  $s(OP)$  for  $d_j$  based on  $k_i$ 
      // Merge point selectivity into selectivity interval
06        $sin = MergeIntoSIN(OP, d_j, s(OP), |k_i|)$ 
07        $SIN[OP, d_j] = sin$ 
08        $SIN[OP].Update(SIN[OP, d_j])$ 
09 return  $SIN$ 

```

Figure 5.40. Computing selectivity intervals.

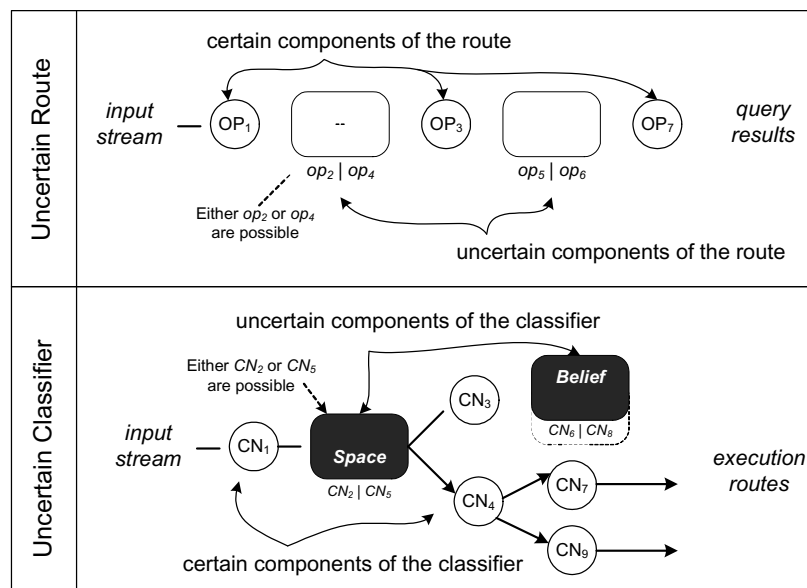


Figure 5.41. Conceptual idea of uncertain routes and classifier.

```

MergeIntoSIN (OP operator,  $d_j$  partition,
s selectivity value, w weight)
01 sin = SIN[OP,  $d_j$ ]
    // if SIN is null, initialize it
02 if (sin == null), then
03     sin.Min = sin.Max = s;
04     return sin;
    // if s is far from the SIN mid-point, return a "conflict"
05 mid = (sin.Max - sin.Min)/2;
06 if ( $\Delta = \text{diff}(s, \text{mid}) > \Delta_{MAX}$ ) then return null;
    // update SIN attributes
07 if ( $s > \text{sin.Max}$ ) then sin.Max = s
08 else if ( $s < \text{sin.Min}$ ) then sin.Min = s
09 sin.PointValue[s] = w
10 return sin

```

Figure 5.42. Merging point estimate into a SIN.

have been ordered, the algorithm determines if any of them are overlapping (Line 3). For all overlapping *SIN*s, *SBF*s are computed (Line 4) as described in Section 5.14.5. After the selectivity belief functions have been established, the algorithm determines the routes by ordering the operators. In every case, where uncertainty intervals are overlapping, i.e., a choice between the operators is uncertain (and there is a corresponding *SBF*), a “*belief space node*” is created in the route (pseudocode is not shown). Figure 5.41 (top) shows a conceptual idea of an uncertain route with belief space nodes.

To complete the uncertainty-aware *QM* solution, the optimizer induces the classifier²⁸ based on the training data tuples and the subsets they belong to (Fig. 5.38, Line 4), which are the certain partitions defined based on the data content.

²⁸The induction algorithm for a “certain classifier” is the same as for to regular decision trees, such as ID3, C4.5 from the literature.

```

ComputeUncertainClassifier ( $T$  training tuples,  $S$  - set of attributes
that may be tested by the decision tree,  $R$  - target route attribute
(predicted by the decision tree),  $M$  - data samples)
01  $Root = \text{DecisionTreeNode}(T)$ 
02 if (all tuples of  $T$  are assigned to the same route  $r_i$ )
03    $Root = \text{single-node tree with label} = r_i$ 
04 else if ( $S$  is empty)
05    $root = \text{single node tree with label} = \text{most common value of } R \text{ in } T$ 
06 else
07    $G \leftarrow \text{members of } S \text{ that maximize } \text{InfoGain}(T, A, M)$ 
08   if ( $|G| > 1$ ) // there are overlapping CINs
09      $BS$  is belief space node  $\forall A \in G$ 
10      $Root.\text{addBeliefSpaceNode}(BS)$ 
11      $Root = BS$ 
12     for (every  $A \in G$ )
13       perform the same steps as in Lines 15-22
14   else if ( $|G| == 1$ ) // there are no overlapping CINs
15      $A \in G$  is decision attribute for  $Root$ 
16     for (each possible value  $v$  of  $A$ )
17       add a branch below  $Root$  testing for  $A = v$ 
18        $T_v \leftarrow \text{subset of } T \text{ with } A = v$ 
19       if ( $T_v$  is empty)
20         below the new branch add a leaf with
           label = most common value of  $R \in T$ 
21       else
           // below the new branch add subtree
22        $Root.\text{addBranch}(\text{ComputeUncertainClassifier}(T_v, S - \{A\}), R)$ 
23 return  $Root$ 

```

Figure 5.43. Computing uncertain classifier.

SIN Computation: Figures 5.40 and 5.42 illustrate the pseudocode describing the details of *SIN* computation. In Figure 5.40, after the *SIN* hashtable that stores selectivity intervals for all operators is initialized (Fig. 5.40, Line 1), for each operator and partition combination, a point selectivity is estimated using every available collected data sample. The computed value for each data sample is then merged with other point estimates to form a selectivity interval for that operator with respect to that partition (Lines 3-6).

Figure 5.42 shows the pseudocode describing how a point estimate is merged with other estimates to form a *SIN* for an operator. First, if the *SIN* is *null*, then the point estimate s becomes the *min* and the *max* value of the *SIN* (Lines 2-4). If s is far from the mid-point of the *SIN*, this is viewed as a “*conflict*” with the current selectivity interval and a *null* value is returned (Lines 5-6). The *conflict flag* (or a null value returned by the procedure) indicates that the selectivity value is significantly different from the selectivity estimates from other samples in that *SIN*. The limit for how far a value may be from the mid-point without causing a conflict is controlled by the system parameter Δ_{MAX} . A significant difference may indicate a possible sample outlier or a change in the environment, thus calling for more samples (representing the latest data) to be collected to be used in *QM* optimization. Otherwise, if s is not conflicting and s is either smaller or larger than the current *min* and *max*, these values are updated accordingly. The point value estimate with its weight, which corresponds to the cardinality of the sample (used in estimating that measure) is stored inside the *SIN* data structure (Lines (7-9)).

Case 2: Uncertain Classifier and Certain Routes This case occurs, when *QM* optimizer uses the *Route-Based* optimization approach (RBA) (see Section 5.14.2 and Figure 5.37(b)). Here routes are computed first, based on all available samples of data and their statistics (Figure 5.38, Line 7). The statistics are considered to be certain, and thus computed routes are assumed to be certain as well. To induce a classifier, the impurity measures for classifying attributes are computed based on all available data samples. Impurity measures for the same attribute may vary for each

data sample, thus leading classification intervals or *CINs*. The presence of *CINs* leads to an uncertain classifier. Figure 5.41 (bottom) shows a conceptual view of an uncertain classifier²⁹.

Computation of Uncertain Classifier: Figure 5.43 shows the pseudocode for constructing uncertain classifier. First, root node is created (Line 1). If all training tuples are assigned to one route, then the classifier is a singleton, or we refer to it “empty” classifier. The understanding here is that all data should be processed using a single route.

Case 3: Uncertain Classifier and Uncertain Routes The third case for multi-route optimization is when both routes and classifier may be uncertain, as a result of merging uncertain *QM* solutions computed using two different optimization approaches (CBA and RBA) – to get a better overall *QM* solution (Figure 5.37(c)).

```

MergeUAQMSolutions (UA-QM1 - CBA-based QM,
UA-QM2 - RBA-based QM)
    // UA-QMfinal is the final (merged) QM
01 UA-QMfinal.C = UA-QM2.C // classifier from RBA-based QM
02 UA-QMfinal.R = MergeRoutes(UA-QM1.R, UA-QM2.R)
03 return UA-QMfinal

```

Figure 5.44. Merging uncertain query meshes.

Merging Uncertainty-Aware QMs: Figure 5.44 shows the pseudocode for merging two query meshes.

5.16 UA-QM Execution

UA-QM Executor receives logical *QM* specification from the optimizer and instantiates physical runtime infrastructure. Conceptually, at runtime, we may have either the classifier with belief space nodes or the routes with belief space nodes which

²⁹ *CN* stands short for “classifier test node”.

represent the uncertainty about the best option (among several possible alternatives). In this section, we describe how these belief space nodes are resolved at runtime to determine the concrete execution solution. In Section 5.16.2 we describe how beliefs are resolved in *UA-QM*.

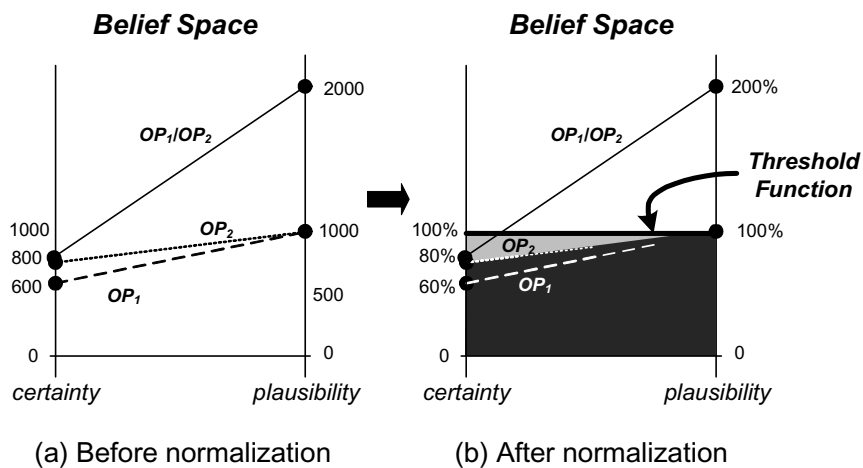


Figure 5.45. Resolving belief functions (route example).

5.16.1 Runtime Infrastructure

For efficient query execution, *QM* Executor uses an infrastructure, called the *Self-Routing Fabric (SRF)* (see Figure 5.31) [27,219], which implements query processing via multiple routes with near-zero route execution overhead. In contrast to current adaptive systems, *SRF* eliminates the expensive central data router operator, such as Eddy operator [33,34,97] and enables de-centralized self-routing of data by operators. Route specifications are encoded in meta-data tuples, called “*routing tokens*” (or short *r-tokens*). *R-tokens* are then embedded inside data streams along with their data tuples by the *online classifier operator*. To keep memory and CPU overheads minimal, the tuples are assigned to an existing route in groups called “*routable clusters*” or short “*rusters*” rather than individual tuples. *Rusters* distinguish themselves from traditional batching, e.g., [33,217], in that they are formed by probing the classifier. Hence, only the tuples that share the same best route get assigned to the same *ruster*.

To enable de-centralized routing, routes in the *r-tokens* are specified in the form of an *operator stack* based on the design of *SRF*. The stack nodes represent the indexes of the operators in the *SRF*, e.g., the *r-token* $\langle 2,3,1,4 \rangle$ indicates that ‘2’ is the first operator in the route, ‘3’ is the next, and so on. A *ruster* is always sent to the operator that is currently the top node in the routing stack. After an operator is done processing the *ruster*, the operator “pops” the top of the routing stack – its unique identifier in the *r-token*, and then puts the *ruster* into the next (now the top) operator’s input queue. When the operator stack is empty, the *ruster* tuples are forwarded to the global output queue reserved by index “0” and then to the application(s).

To enable uncertain route specification, we introduce a small modification to the route encoding: the *r-token* $\langle 2, \{ \mathbf{3} | \mathbf{1} \}, 4 \rangle$ indicates that ‘2’ is the first operator in the route, ‘3’ or ‘1’ is the next, etc. The encoding $\{ 3 | 1 \}$ depicts the uncertainty about the order of operators.

5.16.2 Belief Space Handling

Figure 5.45 visually depicts belief functions using a “relationship graph” (or we denote it as “*belief space*”) where certainty vs. plausibility of each belief are plotted against each other. Thus, resulting BFs represent the uncertainty in the overlapping selectivity sub-intervals.

In this scenario, we have a certain classifier and uncertain routes, i.e., routes with “belief spaces”. In order to determine the concrete physical sequence of operators in an uncertain route, the user receiving results of the query provides a *threshold function* to specify how much he or she is willing to believe in (certainty and possibility of) a relative uncertainty. To make this more intuitive, let’s consider what is the act of believing: it consists of creating a threshold of belief at some probability, assembling evidence for the question, and when the probability exceeds that threshold, accepting it as true. Thus, a threshold function specifies the preference of the user with regard to

uncertainty and symbolically is described as $TF=(C,P)$, where C and P are certainty and possibility parameters.

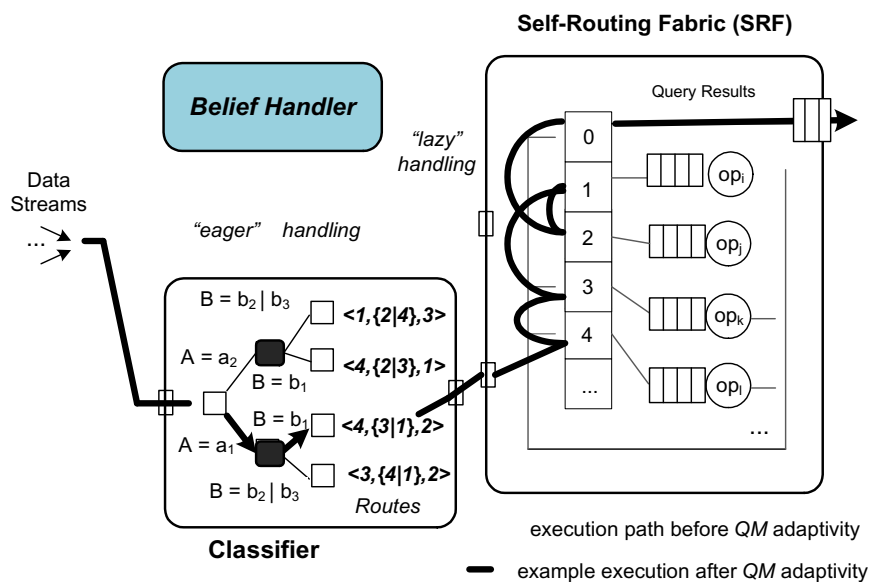
Given the threshold function, a concrete route is determined as follows (see Figure 5.45): for every belief space (depicting relative uncertainty in the order of operators in the route), and the specified threshold function we find the *closest belief function*. If we were to plot belief functions together with the threshold function, we would get something like in Figure 5.45. The threshold function line does not serve as a “cut off” parameter in a traditional sense of a threshold, but rather a desired belief of the user. In order to determine the closest belief function to the threshold function, we employ integration techniques from mathematics (since we need to find a line with the smallest area between that line and the threshold function line).

Definition 5.16.1 (*Smallest Distance Property*) Let A denote area, $y(x) = BF(OP_i)$ represent the belief function of an operator OP_i , and $z(x) = TF()$ represent the threshold function line. Let $A_{OP_i} = \int_{x=0}^{x=1} y(x)dx$, and $A_{TF} = \int_{x=0}^{x=1} z(x)dx$. We choose an operator OP_i if it satisfies the following property: $A_{\Delta} = |A_{TF} - A_{BF}|$ is the smallest \forall BFs in the belief space.

Thus, using mathematical integration we can find the closest line (belief function) to the threshold function, and make a choice among possible alternatives while considering the preferences of the user.

5.17 UA-QM Conclusion

Uncertainty-awareness addresses a major limitation of the most of the existing query processing solutions which typically ignore uncertainty and could result in poor performance, or may lead to frequent re-optimizations, further impeding query performance. Here, we have proposed using the concepts of the belief function theory of evidence can be used as basis for a method that makes multi-route query optimization more robust to uncertainty – both in the plan computation as well as in plan assignment.

Figure 5.46. *UA-QM* execution.

In this paper, we propose to address the challenges above and present a framework handling uncertainty problem in a multi-plan (or multi-route) query execution systems. To be practical, an uncertainty mechanism in a data stream environment must provide: (1) fast measurement of uncertainty, (2) efficient and meaningful representation of different types of uncertainty in a multi-route solution, (3) support for user preferences in uncertainty processing, (4) adaptivity to dynamic changes in uncertainty, and (5) very low overhead compared to traditional continuous query processing. Our uncertainty modeling procedure captures both absolute and relative uncertainty and is compatible with the architecture of existing query optimizers, allowing it to be easily integrated into traditional single-plan-based database management system.

6 CONCLUSION AND FUTURE RESEARCH DIRECTIONS

6.1 Summary

The main goal of this dissertation is to introduce several new features inside Data Stream Management Systems (DSMSs) that are becoming increasingly important requirements for many emerging stream-based applications. Specifically, we tackle the problems of the access control enforcement on streaming data, the tagging of streaming data and the diversity-aware query processing inside DSMSs. The three main contributions of this dissertation can be summarized as follows.

First, the dissertation addresses the problem of continuous access control enforcement in dynamic data stream environments, where both the data and the query security restrictions may potentially change in real-time. The proposed approach advances the state of the art of data stream management systems by introducing: (1) the stream-centric approach to dynamic security, (2) the symmetric security model for both continuous queries and streaming data, and (3) the alternative security-aware query processing methods, that can optimize the execution based on data-related as well as security-related selectivities. Experimental evaluation shows that our proposed approach outperforms other possible alternatives and the security-aware query processing can achieve significant performance benefits over the previously proposed naive pre-filtering and post-filtering methods.

Second, the dissertation proposes a solution for tagging streaming data using a special type of streaming metadata called a tick-tags. Tick-tags can serve a variety of purposes, including labelling or describing some underlying real-time information, and serving as means of disseminating useful knowledge in addition to what is captured by the content of data tuples. Exploiting tags embedded in a data stream increases the kinds of queries that can be executed over data streams. Our experimental results

show the scalability and performance benefits of the tick-tag approach compared to alternative solutions. We have also evaluated the costs of executing tag-aware and tag-oriented continuous queries.

The third contribution in this dissertation is the development of a data diversity-aware query processing framework, called query mesh, that enables different subsets of data to be processed by different query execution plans. We addressed the problem of optimization and efficient runtime execution using query mesh framework, the adaptivity to changing conditions at runtime, and the problem of imprecision and uncertainty in the context of query mesh. The query mesh framework is general, offering numerous advantages over current state-of-the-art solutions. It is applicable to streaming engines and potentially to relational DBMSs as well. Our preliminary experimental results verify the effectiveness of the query mesh approach and demonstrate its potential as a paradigm for continuous query optimization for rich and diverse data.

6.2 Future Research Directions

Next we describe the possible directions for future research based on the concepts presented in this dissertation.

6.2.1 Security Extensions in DSMSs

Security is paramount to the functioning of any system. In this thesis, we have considered only the problem of access control. However, a full-fledged DSMSs needs a comprehensive security support, which involves different issues of security [232]: authentication, authorization and access control, confidentiality and integrity, availability, auditing, privacy, physical, hardware security, and operating system security.

One interesting direction is to expand security punctuation mechanism further to support streaming data integrity, e.g., providing assurance that real-time data traffic is not altered during the transmission. Another area worth examining is the mining

of security preferences (depicted by streaming security punctuations) of the users and building security policy profiles, based on which the best query execution plans can be constructed in advance and anticipated to be used in the future.

6.2.2 Tagging Extensions in DSMSs

In our approach, we tag streaming data using words (strings of characters). An alternative to string-based tags could be object-based tags. An interesting research direction is the investigation whether streaming data could be tagged with objects instead of keywords. Instead of tagging objects with strings, which falls back on a simple full-text search, users could tag something with an actual representation.

Another interesting problem to investigate is whether the *tick-tag* awareness can also be used in query optimization at compile-time when determining a query execution plan, as well as at runtime (similar to *punctuations* [105]) to adapt the query execution strategy based on the observed streaming *tick-tags*. Clearly, not all punctuations are useful to a particular query, and it would be useful to make a determination of when they are. That is, we would like to answer the question “Can stream query Q benefit from a particular set of punctuations?” To that end, we first define punctuation schemes to specify the collection of punctuations that will be presented to a query on a particular data stream. We show how both punctuations and query operators induce groupings over the items in the domain of the input(s). We show that a query benefits from an input punctuation scheme (in terms of being able to produce a given output scheme), if each set in the groupings induced by the operators of the query is covered by a finite number of punctuations in the scheme—a kind of compactness.

Finally, real-time tag mining and tag classification can be of interest to many stream-based applications, e.g., real-time auction monitoring, social networking, and scientific monitoring. This problem refers to extending the support for real-time mining and classification to streaming tags.

6.2.3 Query Mesh Extensions

While so far, this thesis has focused on the core issues of the query mesh optimization and execution, many other topics could potentially be explored in this context.

One interesting direction is to employ learning methods towards different problems inside data stream engine. One such problem in continuous environments is resource limitations problem. Currently there is an underlying assumption that a query mesh produces exact answers under constrained resources. Since query processing eventually may “fail” due to finite resources, alternate solutions may ultimately need to be employed. Here, to reduce the volume of data processed under duress, learning methods may be employed towards adaptive load shedding, results approximation, and disk-spilling. This direction would also help in expanding the understanding of the power and limitations of machine learning techniques in the “guts” of a database engine.

A very important problem is shared multi-query processing using query mesh paradigm. Here, methods for sharing execution routes and classification would need to be investigated not only among subsets of tuples for one query but also among different query meshes for multiple queries executing on the system.

In the long term, it is essential to expand the query mesh scope to consider processing in the context of large-scale distributed environments. Here, issues like resource discovery, business process coordination and resource negotiation must be tackled. Resilience under failures of the network (unreliable communication), servers (fault tolerance), or components of a server (recovery) are critical services, which I would like to explore in my future work.

In both static and streaming databases, a query plan operator typically maintains a *single* queue (see Figure 6.1(a)). Such setup has several disadvantages in the context of multi-route execution: (1) only a single type of scheduling policy can be used by

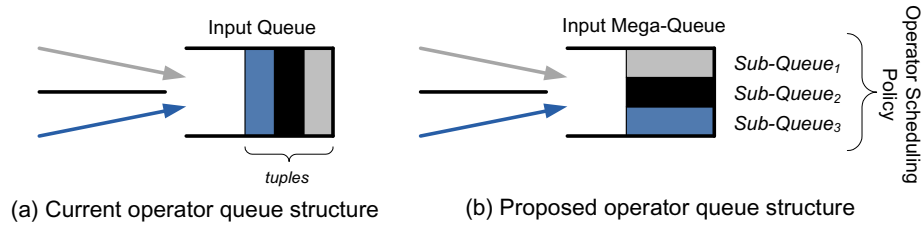


Figure 6.1. Operator queue management.

an operator to dequeue tuples, namely First-In-First-Out (FIFO), and (2) operator processing can be biased towards “bursty” inputs.

Another direction worth examining is the so-called *horizontal queue partitioning* scheme (see Figure 6.1(b)), where the operator input queue is partitioned into multiple sub-queues. Based on horizontally partitioned queues, operators can perform queue management for further improvement for query mesh-based execution. With horizontal queue partitioning approach, there are many possibilities for scheduling tuples for execution by the operators, e.g., FIFO, Priority Queuing, Fair Queuing, Round Robin Weighted Fair Queuing, etc [63]. Other advantages of the proposed operator queue management scheme include: (1) operator can control average queue size, (2) bursts can be absorbed without dropping tuples, (3) operators can prevent bias against “bursty” inputs, and (4) operators can possibly “punish” bursty flows.

The idea of *operator-assisted monitoring* approach, inspired by the resource management approach in *ATM networks* [63] (see Figure 6.2(a)) could potentially also be beneficial in the context of query mesh. The performance metadata, denoted *resource management (RM)* cells, are injected into the network by the intermediate switches and routers and are used to convey network status (e.g., available bandwidth, congestion levels) to the source and destination systems. In the query mesh context, the main idea is for operators to attach their status information, e.g., number of tuples in the queue, current processing rate, etc., to the performance metadata tuples (*p-tokens*) streaming together with the data, so that other operators (e.g., the operators next in route) can become aware of this performance information and possibly exploit

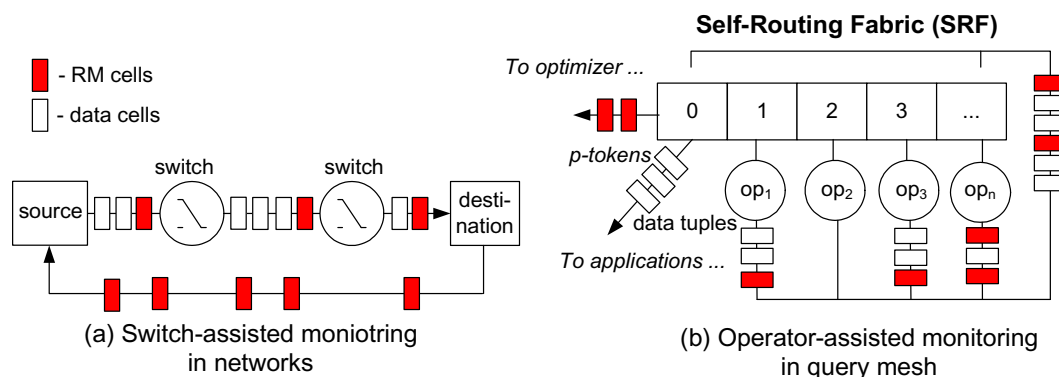


Figure 6.2. Switch and operator-assisted tuning.

it by adapting the routes of the future *rusters* that may contain those bottlenecked operators.

The attractiveness of such operator-assisted performance and state information “diffusion” is in the fact that in a sense it comes for “free”. Since the tuple *rusters* may still have to be routed to other operators, a small “performance signal” with a timestamp can be interleaved with streaming tuples, and then the operators that are left in the route can become “aware” of the current situation at the other operator(s). They can use this information (within a time window) to possibly “detour” some *rusters* if the immediate operators are “overloaded”. Furthermore, these runtime performance metadata can be streamed back to the optimizer to determine how the overall query mesh (not just the runtime routes) can be adapted.

Finally, while our focus in this thesis is on applying diversity-aware query processing in a DSMS, the query mesh model can also be useful in conventional DBMSs, addressing the issue that optimizers sometimes pick plans that perform poorly compared to the actual best plan.

6.2.4 Generalized Punctuation (GPUNCT)

One potentially very effective research direction to explore is the idea of a *generalized punctuation* in DSMSs. To make the idea more intuitive to the reader, consider

the concept of a *Generalized Search Tree (GiST)* [233] in Database Management Systems, which is an index structure supporting an extensible set of queries and data types. The GiST is an extensible data structure, which allows users to develop indices over any kind of data, supporting any lookup over that data. It unifies a number of popular search trees in *one* data structure (the list includes R-trees, B+-trees, hB-trees, TV-trees, Ch-Trees, partial sum trees, ranked B+-trees, and many others), eliminating the need to build multiple search trees for handling diverse applications.

The similar in spirit idea could be extended to the concept of punctuations in Data Stream Management Systems. We refer to this notion – a *generalized punctuation* or *GPUNCT* for short. In a single data structure, the GPUNCT can provide various punctuation logics required by a DSMS, thereby unifying disparate punctuation-based mechanisms. GPUNCT can be used to easily implement a range of well-known punctuation mechanisms, including as sub-stream delimiters inside data streams [98], security metadata [38], feedback mechanisms [103], routing lineage [27], and many others; it can also allow for easy development of specialized metadata for new data types or queries. GPUNCT, like GiST, would represent an example of software extensibility in the context of DSMSs. It will enable the smooth evolution of DSMS towards supporting new punctuation-based algorithms. This would allow authors of new punctuation-based algorithms to focus on implementing the novel features of the new punctuation type – for example, the way in which subsets of the data should be described for search – without becoming experts in DSMS internals.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] Hector Garcia-Molina, Jeffrey D. Ullman, and Jennifer Widom. *Database Systems: The Complete Book*. Prentice Hall, Upper Saddle River, NJ, USA, 2001.
- [2] Lukasz Golab and Tamer Ozsu. Issues in data stream management. *SIGMOD Record*, 32(2):5–14, 2003.
- [3] Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Monitoring streams – A new class of data management applications. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 215–226, 2002.
- [4] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Sailesh Krishnamurthy, Samuel Madden, Vijayshankar Raman, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: Continuous dataflow processing for an uncertain world. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2003.
- [5] Charles D. Cranor, Theodore Johnson, Oliver Spatscheck, and Vladislav Shkapenyuk. Gigascope: A stream database for network applications. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 647–651, 2003.
- [6] Rajeev Motwani, Jennifer Widom, Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Gurmeet Singh Manku, Chris Olston, Justin Rosenstein, and Rohit Varma. Query processing, approximation, and resource management in a data stream management system. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2003.
- [7] Moustafa A. Hammad, Mohamed F. Mokbel, Mohamed H. Ali, Walid G. Aref, Ann Christine Catlin, Ahmed K. Elmagarmid, Mohamed Y. Eltabakh, Mohamed G. Elfeky, Thanaa M. Ghanem, R. Gwadera, Ihab F. Ilyas, Mirette S. Marzouk, and Xiaopeng Xiong. Nile: A query processing engine for data streams. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 851, 2004.
- [8] Elke A. Rundensteiner, Luping Ding, Timothy M. Sutherland, Yali Zhu, Bradford Pielech, and Nishant Mehta. Cape: Continuous query engine with heterogeneous-grained adaptivity. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1353–1356, 2004.
- [9] Jianjun Chen, David J. DeWitt, Feng Tian, and Yuan Wang. NiagaraCQ: A scalable continuous query system for internet databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 379–390, 2000.

- [10] StreamBase. <http://www.streambase.com/>.
- [11] Coral8. <http://www.coral8.com/>.
- [12] Truviso. <http://truviso.com/>.
- [13] Progress Aparma. <http://www.progress.com/apama/index.asp>.
- [14] Andrew Witkowski, Srikanth Bellamkonda, Hua-Gang Li, Vince Liang, Lei Sheng, Wayne Smith, Sankar Subramanian, James Terry, and Tsae-Feng Yu. Continuous queries in Oracle. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1173–1184, 2007.
- [15] My Heart. <http://www.hitech-projects.com/euprojects/myheart/>.
- [16] PIPS. <http://www.pips.eu.org/>.
- [17] Proactive Health. <http://www.intel.com/research/prohealth/>.
- [18] Bright Kite. <http://brightkite.com/>.
- [19] The Carbon Project. <http://www.thecarbonproject.com/>.
- [20] S. Christodoulakis. Implications of certain assumptions in database performance evaluation. *ACM Transactions on Database Systems (TODS)*, 9(2):163–186, 1984.
- [21] Daniel Lewis. What is Web 2.0? *Crossroads*, 13(1):3–3, 2006.
- [22] Hak Lae Kim, John G. Breslin, Sung-Kwon Yang, and Hong-Gee Kim. Social semantic cloud of tag: Semantic model for social tagging. In *Proceedings of the Agent and Multi-Agent Systems: Technologies and Applications (KES-AMSTA)*, pages 83–92, 2008.
- [23] Liming Chen and Craig Roberts. Semantic tagging for large-scale content management. In *Proceedings of the Web Intelligence*, pages 478–481, 2007.
- [24] Pirjo Näkki, Sari Vainikainen, and Asta Bäck. Experiences of semantic tagging with Tilkut. In *Proceedings of the International Conference on Entertainment and Media in the Ubiquitous Era (Mindtrek)*, pages 167–171, 2008.
- [25] Simone Braun, Valentin Zacharias, and Hans-Jörg Happel. Social semantic bookmarking. In *Proceedings of the Practical Aspects of Knowledge Management (PAKM)*, pages 62–73, 2008.
- [26] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill Higher Education, New York, NY, USA, 2000.
- [27] Rimma V. Nehme, Elke A. Rundensteiner, and Elisa Bertino. Self-tuning query mesh for adaptive multi-route query processing. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 803–814, 2009.
- [28] Pedro Bizarro, Shivnath Babu, David J. DeWitt, and Jennifer Widom. Content-based routing: Different plans for different data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 757–768, 2005.

- [29] Neoklis Polyzotis. Selectivity-based partitioning: A divide-and-union paradigm for effective query optimization. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 720–727, 2005.
- [30] Microsoft SQL Server. <http://www.microsoft.com/sql/default.mspx>.
- [31] DB2. <http://www.ibm.com/software/data/db2/>.
- [32] Oracle. <http://www.oracle.com/index.html>.
- [33] Amol Deshpande. An initial study of overheads of eddies. *SIGMOD Record*, 33(1):44–49, 2004.
- [34] Ron Avnur and Joseph M. Hellerstein. Eddies: Continuously adaptive query processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 261–272. ACM, 2000.
- [35] W. Lindner and J. Meier. Towards a secure data stream management system. In *Proceedings of the Trends in Enterprise Application Architecture (TEAA)*, pages 114–128, 2005.
- [36] Wolfgang Lindner and Jörg Meier. Securing the borealis data stream engine. In *Proceedings of the International Database Engineering and Applications Symposium (IDEAS)*, pages 137–147, 2006.
- [37] Barbara Carminati, Elena Ferrari, and Kian Lee Tan. Enforcing access control over data streams. In *Proceedings of the Symposium on Access Control Models and Technologies (SACMAT)*, pages 21–30, 2007.
- [38] Rimma V. Nehme, Elke A. Rundensteiner, and Elisa Bertino. A security punctuation framework for enforcing access control on streaming data. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 406–415, 2008.
- [39] Rimma V. Nehme, Hyo-Sang Lim, and Elisa Bertino. Fence: Continuous access control enforcement in dynamic data stream environments. Submitted to the International Conference on Very Large Data Bases (VLDB), 2009.
- [40] Rimma V. Nehme, Elke A. Rundensteiner, and Elisa Bertino. Tagging stream data for rich real-time services. Submitted to the International Conference on Very Large Data Bases (VLDB), 2009.
- [41] Rimma V. Nehme, Karen E. Works, Elke A. Rundensteiner, and Elisa Bertino. Query mesh: Multi-route query processing technology. Submitted to the International Conference on Very Large Data Bases (VLDB), 2009.
- [42] Rimma V. Nehme, Hyo-Sang Lim, Elisa Bertino, and Elke A. Rundensteiner. Streamshield: A stream-centric approach towards security and privacy in data stream environments. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2009.
- [43] Arvind Arasu, Brian Babcock, Shivnath Babu, Mayur Datar, Keith Ito, Itaru Nishizawa, Justin Rosenstein, and Jennifer Widom. STREAM: The stanford stream data manager. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, page 665, 2003.

- [44] Magdalena Balazinska, Hari Balakrishnan, and Michael Stonebraker. Contract-based load management in federated distributed systems. In *Proceedings of the Conference on Symposium on Networked Systems Design and Implementation*, pages 197–210, 2004.
- [45] The PostgreSQL object relational database management system. <http://www.postgresql.org>, 2009.
- [46] Sailesh Krishnamurthy, Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Frederick Reiss, and Mehul A. Shah. TelegraphCQ: An architectural status report. *IEEE Data Engineering Bulletin*, 26(1):11–18, 2003.
- [47] Thanana M. Ghanem. Supporting predicate-window queries in data stream management systems. In *Proceedings of the International Conference on Data Engineering Workshops*, page 139, 2006.
- [48] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, 15(2):121–142, 2006.
- [49] Mehmet Altinel and Michael J. Franklin. Efficient filtering of xml documents for selective dissemination of information. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 53–64, 2000.
- [50] Yanlei Diao and Michael J. Franklin. Query processing for high-volume xml message brokering. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 261–272, 2003.
- [51] Benjamin Nguyen, Serge Abiteboul, Gregory Cobena, and Mihai Preda. Monitoring xml data on the web. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 437–448, 2001.
- [52] Ulf Schreier, Hamid Pirahesh, Rakesh Agrawal, and C. Mohan. Alert: An architecture for transforming a passive DBMS into an active DBMS. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 469–478, 1991.
- [53] Douglas B. Terry, David Goldberg, David A. Nichols, and Brian M. Oki. Continuous queries over append-only databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 321–330, 1992.
- [54] Ling Liu, Calton Pu, and Wei Tang. Continual queries for internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 11(4):610–628, 1999.
- [55] Ashish Gupta and Inderpal Singh Mumick. Maintenance of materialized views: Problems, techniques, and applications. *IEEE Data Engineering Bulletin*, 18(2):3–18, 1995.
- [56] Jennifer Widom and Stefano Ceri, editors. *Active Database Systems: Triggers and Rules For Advanced Database Processing*. Morgan Kaufmann, San Francisco, CA, USA, 1996.

- [57] Shivnath Babu. *Adaptive Query Processing in Data Stream Management Systems*. PhD thesis, Stanford University, 2005.
- [58] Shivnath Babu, Rajeev Motwani, Kamesh Munagala, Itaru Nishizawa, and Jennifer Widom. Adaptive ordering of pipelined stream filters. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 407–418, 2004.
- [59] Yannis E. Ioannidis and Younkyung Cha Kang. Left-deep vs. bushy trees: An analysis of strategy spaces and its implications for query optimization. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 168–177, 1991.
- [60] Ravi Krishnamurthy, Haran Boral, and Carlo Zaniolo. Optimization of nonrecursive queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 128–137, 1986.
- [61] Arun N. Swami and Balakrishna R. Iyer. A polynomial time algorithm for optimizing join queries. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 345–354, 1993.
- [62] Stratis Viglas, Jeffrey F. Naughton, and Josef Burger. Maximizing the output rate of multi-way join queries over streaming information sources. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 285–296, 2003.
- [63] James F. Kurose and Keith Ross. *Computer Networking: A Top-Down Approach*. Addison-Wesley, Boston, MA, USA, 2002.
- [64] Amol Deshpande and Joseph M. Hellerstein. Lifting the burden of history from adaptive query processing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 948–959, 2004.
- [65] Vijayshankar Raman, Amol Deshpande, and Joseph M. Hellerstein. Using state modules for adaptive query processing. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 353–365, 2003.
- [66] Goetz Graefe and William J. McKenna. The Volcano optimizer generator: Extensibility and efficient search. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 209–218, 1993.
- [67] Morton M. Astrahan, Mike W. Blasgen, Donald D. Chamberlin, Kapali P. Eswaran, Jim Gray, Patricia P. Griffiths, W. Frank King III, Raymond A. Lorie, Paul R. McJones, James W. Mehl, Gianfranco R. Putzolu, Irving L. Traiger, Bradford W. Wade, and Vera Watson. System R: Relational approach to database management. *ACM Transactions on Database Systems (TODS)*, 1(2):97–137, 1976.
- [68] Zachary G. Ives, Alon Y. Halevy, and Daniel S. Weld. Adapting to source properties in processing data integration queries. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 395–406, 2004.
- [69] Paul De Bra and Jan Paredaens. Horizontal decompositions and their impact on query solving. *SIGMOD Record*, 13(1):46–50, 1982.

- [70] Amol Deshpande, Carlos Guestrin, Wei Hong, and Samuel Madden. Exploiting correlated attributes in acquisitional query processing. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 143–154, 2005.
- [71] Chung-Min Chen and Nick Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 161–172, 1994.
- [72] Quanzhong Li, Minglong Shao, Volker Markl, Kevin S. Beyer, Latha S. Colby, and Guy M. Lohman. Adaptively reordering joins during query execution. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 26–35, 2007.
- [73] Michael Stillger, Guy M. Lohman, Volker Markl, and Mokhtar Kandil. LEO - DB2's LEarning Optimizer. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 19–28, 2001.
- [74] Volker Markl and Guy Lohman. Learning table access cardinalities with LEO. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 613–613, 2002.
- [75] Ning Zhang 0002, Peter J. Haas, Vanja Josifovski, Guy M. Lohman, and Chun Zhang. Statistical learning techniques for costing xml queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 289–300, 2005.
- [76] Amol Deshpande, Zachary G. Ives, and Vijayshankar Raman. Adaptive query processing. *Foundations and Trends in Databases*, 1(1):1–140, 2007.
- [77] Brian Babcock and Surajit Chaudhuri. Towards a robust query optimizer: A principled and practical approach. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 119–130, 2005.
- [78] Shivnath Babu, Pedro Bizarro, and David J. DeWitt. Proactive re-optimization. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 107–118, 2005.
- [79] Volker Markl, Vijayshankar Raman, David E. Simmen, Guy M. Lohman, and Hamid Pirahesh. Robust query processing through progressive optimization. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 659–670, 2004.
- [80] Efstratios Viglas. *Novel query optimization and evaluation techniques*. PhD thesis, University of Wisconsin – Madison, 2003.
- [81] Francis C. Chu, Joseph Y. Halpern, and Johannes Gehrke. Least expected cost query optimization: What can we expect? In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 293–302, 2002.
- [82] Yannis E. Ioannidis, Raymond T. Ng, Kyuseok Shim, and Timos K. Sellis. Parametric query optimization. *VLDB Journal*, 6(2):132–151, 1997.
- [83] Norbert Fuhr. A probabilistic framework for vague queries and imprecise information in db. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 696–707, 1990.

- [84] G. Grahne. *Problem of Incomplete Information in Relational Databases*. Springer-Verlag, New York, NY, USA, 1991.
- [85] Lyublena Antova, Christoph Koch, and Dan Olteanu. Query language support for incomplete information in the maybms system. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1422–1425, 2007.
- [86] Omar Benjelloun, Anish Das Sarma, Alon Y. Halevy, Martin Theobald, and Jennifer Widom. Databases with uncertainty and lineage. *VLDB Journal*, 17(2):243–264, 2008.
- [87] Laks V. S. Lakshmanan, Nicola Leone, Robert B. Ross, and V. S. Subrahmanian. Probview: A flexible probabilistic database system. *ACM Transactions on Database Systems (TODS)*, 22(3):419–469, 1997.
- [88] Jihad Boulos, Nilesh N. Dalvi, Bhushan Mandhani, Shobhit Mathur, Christopher Ré, and Dan Suciu. MYSTIQ: A system for finding more answers by using probabilities. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 891–893, 2005.
- [89] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 262–276, 2005.
- [90] Shivnath Babu and Pedro Bizarro. Adaptive query processing in the looking glass. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 238–249, 2005.
- [91] Nicolas Bruno and Surajit Chaudhuri. Exploiting statistics on query expressions for optimization. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 263–274, 2002.
- [92] Zachary G. Ives, Daniela Florescu, Marc Friedman, Alon Y. Levy, and Daniel S. Weld. An adaptive query execution system for data integration. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 299–310, 1999.
- [93] Navin Kabra and David J. DeWitt. Efficient mid-query re-optimization of sub-optimal query execution plans. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 106–117, 1998.
- [94] Zachary George Ives. *Efficient query processing for data integration*. PhD thesis, University of Washington, 2002.
- [95] Tolga Urhan, Michael J. Franklin, and Laurent Amsaleg. Cost based query scrambling for initial delays. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 130–141, 1998.
- [96] Samuel Madden, Mehul A. Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 49–60, 2002.

- [97] Feng Tian and David J. DeWitt. Tuple routing strategies for distributed eddies. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 333–344, 2003.
- [98] Peter A. Tucker, David Maier, and Tim Sheard. Applying punctuation schemes to queries over continuous data streams. *IEEE Data Engineering Bulletin*, 26(1):33–40, 2003.
- [99] Luping Ding, Nishant Mehta, Elke A. Rundensteiner, and George T. Heineman. Joining punctuated streams. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 587–604, 2004.
- [100] Luping Ding and Elke A. Rundensteiner. Evaluating window joins over punctuated streams. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 98–107, 2004.
- [101] Leonidas Fegaras, David Levine, Sujoe Bose, and Vamsi Chaluvadi. Query processing of streamed xml data. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 126–133, 2002.
- [102] Hua-Gang Li, Songting Chen, Junichi Tatemura, Divyakant Agrawal, K. Selçuk Candan, and Wang-Pin Hsiung. Safety guarantee of continuous join queries over punctuated data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 19–30, 2006.
- [103] Rafael J. Fernandez-Moctezuma, Kristin A. Tufte, and Jin Li. Inter-operator feedback in data stream management. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2009.
- [104] Peter Tucker. *Punctuated data streams*. PhD thesis, OGI School of Science & Engineering at Oregon Health and Science University, 2005.
- [105] Peter A. Tucker, David Maier, Tim Sheard, and Leonidas Fegaras. Exploiting punctuation semantics in continuous data streams. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):555–568, 2003.
- [106] Theodore Johnson, S. Muthukrishnan, Vladislav Shkapenyuk, and Oliver Spatscheck. A heartbeat mechanism and its application in Gigascope. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1079–1088, 2005.
- [107] Charu C. Aggarwal, Jiawei Han, Jianyong Wang, and Philip S. Yu. On demand classification of data streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 503–508, 2004.
- [108] Sudipto Guha, Nina Mishra, Rajeev Motwani, and Liadan O’Callaghan. Clustering data streams. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 359–366, 2000.
- [109] Geoff Hulten, Laurie Spencer, and Pedro Domingos. Mining time-changing data streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 97–106, 2001.
- [110] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 71–80, 2000.

- [111] Nick Koudas, Beng Chin Ooi, Kian-Lee Tan, and Rui Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 804–815, 2004.
- [112] Graham Cormode, Theodore Johnson, Flip Korn, S. Muthukrishnan, Oliver Spatscheck, and Divesh Srivastava. Holistic udafs at streaming speeds. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 35–46, 2004.
- [113] Graham Cormode, Flip Korn, S. Muthukrishnan, and Divesh Srivastava. Finding hierarchical heavy hitters in data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 464–475, 2003.
- [114] Jeffrey Scott Vitter. Random sampling with a reservoir. *ACM Transactions in Mathematics Software*, 11(1):37–57, 1985.
- [115] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 635–644, 2002.
- [116] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 286–296, 2004.
- [117] Moses Charikar, Liadan O’Callaghan, and Rina Panigrahy. Better streaming algorithms for clustering problems. In *Proceedings of the Symposium on the Theory of Computing (STOC)*, pages 30–39, 2003.
- [118] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O’Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 15(3):515–528, 2003.
- [119] João Gama, Ricardo Rocha, and Pedro Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 523–528, 2003.
- [120] Graham Cormode and S. Muthukrishnan. What’s hot and what’s not: Tracking most frequent items dynamically. *ACM Transactions on Database Systems (TODS)*, 30(1):249–278, 2005.
- [121] Pierre-Alain Laur, Richard Nock, Jean-Emile Symphor, and Pascal Poncelet. Mining evolving data streams for frequent patterns. *Pattern Recognition*, 40(2):492–503, 2007.
- [122] Yixin Chen, Guozhu Dong, Jiawei Han, Benjamin W. Wah, and Jianyong Wang. Multi-dimensional regression analysis of time-series data streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 323–334, 2002.
- [123] Valery Guralnik and Jaideep Srivastava. Event detection from time series data. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 33–42, 1999.

- [124] Johannes Gehrke, Venkatesh Ganti, Raghu Ramakrishnan, and Wei-Yin Loh. Boat – optimistic decision tree construction. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 169–180, 1999.
- [125] Johannes Gehrke, Raghu Ramakrishnan, and Venkatesh Ganti. Rainforest – A framework for fast decision tree construction of large datasets. *Data Mining Knowledge Discovery*, 4(2-3):127–162, 2000.
- [126] Manish Mehta, Rakesh Agrawal, and Jorma Rissanen. SLIQ: A fast scalable classifier for data mining. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 18–32, 1996.
- [127] John C. Shafer, Rakesh Agrawal, and Manish Mehta. SPRINT: A scalable parallel classifier for data mining. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 544–555, 1996.
- [128] Mohamed Medhat Gaber, Arkady Zaslavsky, and Shonali Krishnaswamy. Mining data streams: A review. *SIGMOD Record*, 34(2):18–26, 2005.
- [129] Ralf Klinkenberg. Learning drifting concepts: Example selection vs. example weighting. *Intelligent Data Analysis*, 8(3):281–300, 2004.
- [130] Miroslav Kubat and Gerhard Widmer. Adapting to drift in continuous domains (Extended Abstract). In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 307–310, 1995.
- [131] Gerhard Widmer and Miroslav Kubat. Learning in the presence of concept drift and hidden contexts. *Machine Learning*, 23(1):69–101, 1996.
- [132] Erich Schikuta and Paul Glantschnig. An object-relational neural network database type. In *Artificial Intelligence and Applications*, pages 31–37, 2007.
- [133] José Cascalho and Helder Coelho. Tuning agents’ behaviours using embedded attributes. In *Artificial Intelligence and Applications*, pages 157–162, 2007.
- [134] Stepas Janusonis and Liudas Leonas. Future trends in self-formation. In *Artificial Intelligence and Applications*, pages 627–632, 2005.
- [135] David Gilbert and Christopher J. Hogger. Logic for representing and implementing knowledge about system behaviour. In *Advanced Topics in Artificial Intelligence*, pages 42–49, 1992.
- [136] Jirí Lazanský. Practical applications of planning tasks. In *Advanced Topics in Artificial Intelligence*, pages 238–244, 1992.
- [137] Menno Heeren. Ant system for solving reactive scheduling problems in value added chains. In *Artificial Intelligence and Applications*, pages 6–11, 2005.
- [138] Ivan Stajduhar and Ivan Bratko. Likelihood based classification in bayesian networks. In *Artificial Intelligence and Applications*, pages 367–372, 2007.
- [139] Richard J. Povinelli. Book review: Foundations of genetic programming. *Genetic Programming and Evolvable Machines*, 5(3):319–320, 2004.
- [140] Durga Shrestha. Machine learning approaches for estimation of prediction interval for the model output. *Neural Networks*, 19(2):225–235, 2006.

- [141] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, USA.
- [142] Ilyes Jenhani, Nahla Ben Amor, and Zied Elouedi. Decision trees as possibilistic classifiers. *International Journal of Approximate Reasoning*, 48(3):784–807, 2008.
- [143] Siegfried Nijssen. Bayes optimal classification for decision trees. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 696–703, 2008.
- [144] Nick Street and YongSeog Kim. A streaming ensemble algorithm (SEA) for large-scale classification. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 377–382, 2001.
- [145] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Hippocratic databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 143–154, 2002.
- [146] Carlos Ribeiro, Andre Zuquete, Paulo Ferreira, and Paulo Guedes. SPL: An access control language for security policies and complex constraints. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2001.
- [147] Piero A. Bonatti, Sabrina De Capitani di Vimercati, and Pierangela Samarati. An algebra for composing access control policies. *ACM Transactions on Information and System Security (TISSEC)*, 5(1):1–35, 2002.
- [148] Dominik Raub and Rainer Steinwandt. An algebra for enterprise privacy policies closed under composition and conjunction. In *Proceedings of the International Conference on Emerging Trends in Information and Communication Security (ETRICS)*, pages 130–144, 2006.
- [149] Elisa Bertino, Piero A. Bonatti, and Elena Ferrari. TRBAC: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):191–233, 2001.
- [150] Maria Luisa Damiani, Elisa Bertino, Barbara Catania, and Paolo Perlasca. GEO-RBAC: A spatially aware RBAC. *ACM Transactions on Information and System Security (TISSEC)*, 10(1), 2007.
- [151] Jaehong Park and Ravi Sandhu. Towards usage control models: Beyond traditional access control. In *Proceedings of the Symposium on Access Control Models and Technologies (SACMAT)*, pages 57–64, 2002.
- [152] Jaehong Park and Ravi Sandhu. The UCON-ABC usage control model. *ACM Transactions on Information and System Security (TISSEC)*, 7(1):128–174, 2004.
- [153] Qihua Wang, Ting Yu, Ninghui Li, Jorge Lobo, Elisa Bertino, Keith Irwin, and Ji-Won Byun. On the correctness criteria of fine-grained access control in relational databases. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 555–566, 2007.
- [154] Surajit Chaudhuri, Tanmoy Dutta, and S. Sudarshan. Fine grained authorization through predicated grants. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 1174–1183, 2007.

- [155] Govind Kabra, Ravishankar Ramamurthy, and S. Sudarshan. Redundancy and information leakage in fine-grained access control. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 133–144, 2006.
- [156] Arun Kumar, Neeran M. Karnik, and Girish Chaffle. Context sensitivity in role-based access control. *Operating Systems Review*, 36(3):53–66, 2002.
- [157] Deepavali Bhagwat, Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya. An annotation management system for relational databases. *VLDB Journal*, 14(4):373–396, 2005.
- [158] Floris Geerts, Anastasios Kementsietsidis, and Diego Milano. Mondrian: Annotating and querying databases through colors and blocks. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 82, 2006.
- [159] Mohamed Y. Eltabakh, Mourad Ouzzani, and Walid G. Aref. bdbms – A database management system for biological data. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, pages 196–206, 2007.
- [160] Divesh Srivastava and Yannis Velegrakis. Intensional associations between data and metadata. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 401–412, 2007.
- [161] Flickr. <http://flickr.com/>, 2009.
- [162] Delicious. <http://del.icio.us/>.
- [163] Technorati. <http://www.technorati.com/>.
- [164] Cameron Marlow, Mor Naaman, Danah Boyd, and Marc Davis. HT06, tagging paper, taxonomy, Flickr, academic article, to read. In *Proceedings of the Conference on Hypertext and Hypermedia (Hypertext)*, pages 31–40, 2006.
- [165] Ed H. Chi and Todd Mytkowicz. Understanding the efficiency of social tagging systems using information theory. In *Proceedings of the Conference on Hypertext and Hypermedia (Hypertext)*, pages 81–88, 2008.
- [166] Harry Halpin, Valentin Robu, and Hana Shepherd. The complex dynamics of collaborative tagging. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 211–220, 2007.
- [167] Christoph Schmitz, Andreas Hotho, Robert Jschke, and Gerd Stumme. Mining association rules in folksonomies. In *Proceedings of the International Federation of Classification Societies (IFCS)*, pages 261–270, Heidelberg, 2006.
- [168] Paul Heymann, Daniel Ramage, and Hector Garcia-Molina. Social tag prediction. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR)*, pages 531–538, 2008.
- [169] Steffen Oldenburg, Martin Garbe, and Clemens H. Cap. Similarity cross-analysis of tag/co-tag spaces in social classification systems. In *Proceeding of the ACM Workshop on Search in Social Media (SSM)*, pages 11–18, 2008.
- [170] Dan Olteanu, Tim Furche, and François Bry. An efficient single-pass query evaluator for XML data streams. In *Proceedings of the Symposium on Applied computing (SAC)*, pages 627–631, 2004.

- [171] Feng Peng and Sudarshan S. Chawathe. XPath queries on streaming data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 431–442, 2003.
- [172] Xiaogang Li and Gagan Agrawal. Efficient evaluation of XQuery over streaming data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 265–276, 2005.
- [173] Eugene Wu, Yanlei Diao, and Shariq Rizvi. High-performance complex event processing over streams. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 407–418, 2006.
- [174] Mohamed H. Ali, Walid G. Aref, Raja Bose, Ahmed K. Elmagarmid, Abdel-salam Helal, Ibrahim Kamel, and Mohamed F. Mokbel. NILE-PDT: A phenomenon detection and tracking framework for data stream management systems. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1295–1298, 2005.
- [175] David F. Ferraiolo, Ravi S. Sandhu, Serban I. Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4(3):224–274, 2001.
- [176] Daniel J. Abadi, Donald Carney, Ugur Çetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Michael Stonebraker, Nesime Tatbul, and Stanley B. Zdonik. Aurora: A new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.
- [177] Elisa Bertino, Pierangela Samarati, and Sushil Jajodia. An extended authorization model for relational databases. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 9(1):85–101, 1997.
- [178] Duminda Wijesekera and Sushil Jajodia. A propositional policy algebra for access control. *ACM Transactions on Information and System Security (TISSEC)*, 6(2):286–325, 2003.
- [179] Wenfei Fan, Chee-Yong Chan, and Minos Garofalakis. Secure xml querying with security views. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 587–598, 2004.
- [180] Shariq Rizvi, Alberto O. Mendelzon, S. Sudarshan, and Prasan Roy. Extending query rewriting techniques for fine-grained access control. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 551–562, 2004.
- [181] Thomas Brinkhoff. Generating network-based moving objects. In *Proceedings of the Statistical and Scientific Database Management (SSDBM)*, page 253, 2000.
- [182] Jin Li, Kristin Tufte, Vladislav Shkapenyuk, Vassilis Papadimos, Theodore Johnson, and David Maier. Out-of-order processing: A new architecture for high-performance stream systems. *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1(1):274–288, 2008.

- [183] Luping Ding and Elke A. Rundensteiner. Evaluating window joins over punctuated streams. In *Proceedings of the International Conference on Information and Knowledge Management (CIKM)*, pages 98–107, 2004.
- [184] Christoph Koch, Stefanie Scherzinger, Nicole Schweikardt, and Bernhard Stegmaier. FluXQuery: An optimizing XQuery processor for streaming xml data. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 1309–1312, 2004.
- [185] Sujoe Bose and Leonidas Fegaras. Data stream management for historical xml data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 239–250, 2004.
- [186] Hong Su, Elke A. Rundensteiner, and Murali Mani. Semantic query optimization for XQuery over xml streams. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 277–288, 2005.
- [187] Bettina Berendt and Christoph Hanser. Tags are not metadata, but “just more content” – to some people. In *Proceedings of the International Conference on Weblogs and Social Media (ICWSM)*, 2007.
- [188] Ronen Feldman, Binyamin Rosenfeld, Moshe Fresko, and Brian D. Davison. Hybrid semantic tagging for information extraction. In *Proceedings of the International World Wide Web Conference (WWW)*, pages 1022–1023, 2005.
- [189] Keith Ellul, Bryan Krawetz, Jeffrey Shallit, and Ming-Wei Wang. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics*, 10(4):407–437, 2005.
- [190] Morgan Ames and Mor Naaman. Why we tag: Motivations for annotation in mobile and online media. In *Proceedings of the Conference on Computer Human Interaction (CHI)*, pages 971–980, 2007.
- [191] Scott A. Golder and Bernardo A. Huberman. The structure of collaborative tagging systems. *Proceedings of the Computing Research Repository (CoRR)*, 2005.
- [192] Nathan Eagle and Alex Pentland. Reality mining: Sensing complex social systems. *Personal and Ubiquitous Computing (PUC)*, 10(4):255–268, 2006.
- [193] Robert Jäschke, Leandro Balby Marinho, Andreas Hotho, Lars Schmidt-Thieme, and Gerd Stumme. Tag recommendations in social bookmarking systems. *Artificial Intelligence Communications*, 21(4):231–247, 2008.
- [194] EBay. <http://www.ebay.com/>.
- [195] Wen-Liang Hung and Miin-Shen Yang. Similarity measures between type-2 fuzzy sets. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 12(6):827–842, 2004.
- [196] Many Eyes DS. <http://manyeyes.alphaworks.ibm.com/>.
- [197] Google Latitude. <http://www.google.com/latitude/intro.html>.
- [198] TradingMarkets. <http://www.tradingmarkets.com/>.

- [199] Trending. <http://www.trending123.com/>.
- [200] Richard J. Lipton, Jeffrey F. Naughton, Donovan A. Schneider, and S. Seshadri. Efficient sampling strategies for relational database operations. *Theoretical Computer Science*, 116(1&2):195–226, 1993.
- [201] David J. Hand, Padhraic Smyth, and Heikki Mannila. *Principles of data mining*. MIT Press, Cambridge, MA, USA, 2001.
- [202] Dymitr Ruta. Dynamic data condensation for classification. In *Proceedings of the Artificial Intelligence and Soft Computing (ICAISC)*, pages 672–681, 2006.
- [203] B.V. Dasarathy. Minimal consistent set identification for optimal NN decision systems. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 24(3):511–517, 1994.
- [204] Takekazu Kato and Toshikazu Wada. Direct condensing: An efficient voronoi condensing algorithm for nearest neighbor classifiers. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, pages 474–477, 2004.
- [205] C. H. Chen and Adam Józwiak. A sample set condensation algorithm for the class sensitive artificial neural network. *Pattern Recognition*, 17(8):819–823, 1996.
- [206] José Salvador Sánchez. High training set size reduction by space partitioning and prototype abstraction. *Pattern Recognition*, 37(7):1561–1564, 2004.
- [207] Maria Teresa Lozano. *Data Reduction Techniques in Classification Processes*. PhD thesis, Jaume University, 2007.
- [208] Martin Klazar. Bell numbers, their relatives, and algebraic differential equations. *Journal of Combinatorial Theory Series*, 102(1):63–87, 2003.
- [209] Surajit Chaudhuri. An overview of query optimization in relational systems. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 34–43, 1998.
- [210] Patricia G. Selinger, Morton M. Astrahan, Donald D. Chamberlin, Raymond A. Lorie, and Thomas G. Price. Access path selection in a relational database management system. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 23–34, 1979.
- [211] Joseph M. Hellerstein and Michael Stonebraker. Predicate migration: Optimizing queries with expensive predicates. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 267–276, 1993.
- [212] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, New York, NY, USA, 1990.
- [213] Yannis E. Ioannidis and Younkyung Cha Kang. Randomized algorithms for optimizing large join queries. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 312–321, 1990.
- [214] Arvind Arasu, Brian Babcock, Shivnath Babu, Jon McAlister, and Jennifer Widom. Characterizing memory requirements for queries over continuous data streams. *ACM Transactions on Database Systems (TODS)*, 29:162–194, 2004.

- [215] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proceedings of the Symposium on Principles of Database Systems (PODS)*, pages 1–16, 2002.
- [216] Stratis Viglas and Jeffrey F. Naughton. Rate-based query optimization for streaming information sources. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 37–48, 2002.
- [217] Kajal T. Claypool and Mark Claypool. Teddies: Trained eddies for reactive stream processing. In *Proceedings of the International Conference on Database Systems for Advanced Applications (DASFAA)*, pages 220–234, 2008.
- [218] Kien A. Hua, Yu lung Lo, and Honesty C. Young. Considering data skew factor in multi-way join query optimization for parallel execution. *VLDB Journal*, 2(3):303–330, 1993.
- [219] Rimma V. Nehme, Elke A. Rundensteiner, Karen E. Works, and Elisa Bertino. Query mesh: An efficient multi-route approach to query optimization. Technical Report CSD TR #08-009, Department of Computer Science, Purdue University, April 2008.
- [220] Gerhard Widmer and Miroslav Kubat. Effective learning in dynamic environments by explicit context tracking. In *Proceedings of the European Conference on Machine Learning (ECML)*, pages 227–243, 1993.
- [221] Alexey Tsymbal. The problem of concept drift: Definitions and related work. Technical Report TCD-CS-2004-15, Department of Computer Science, University of Dublin, Trinity College, 2004.
- [222] Surajit Chaudhuri, Arnd Christian König, and Vivek R. Narasayya. Sqlcm: A continuous monitoring framework for relational database engines. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 473–485, 2004.
- [223] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, Upper Saddle River, NJ, USA, 1994.
- [224] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.
- [225] Haixun Wang and Jian Pei. A random method for quantifying changing distributions in data streams. In *Proceedings of the Principles of Data Mining and Knowledge Discovery (PKDD)*, pages 684–691, 2005.
- [226] Ziv Bar-Yossef, T. S. Jayram, Robert Krauthgamer, and Ravi Kumar. Approximating edit distance efficiently. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 550–559, 2004.
- [227] I. M. Chakravarti, R. G. Laha, and J. Roy. *Handbook of Methods of Applied Statistics*, volume I. John Wiley and Sons, New York, NY, USA, 1967.
- [228] A. Kumaran, Pavan K. Chowdary, and Jayant R. Haritsa. On pushing multilingual query operators into relational engines. In *Proceedings of the International Conference on Data Engineering (ICDE)*, page 98, 2006.

- [229] Ian H. Witten and Eibe Frank. Data mining: Practical machine learning tools and techniques with Java implementations. *In SIGMOD Record*, 31(1):76–77, 2002.
- [230] Philippe Smets. The transferable belief model. *Artificial Intelligence*, 66(2):191–234, 1994.
- [231] Philippe Smets. The application of the transferable belief model to diagnostic problems. *International journal of intelligent systems*, 13:127–157, 1998.
- [232] Matthew A. Bishop. *The Art and Science of Computer Security*. Addison-Wesley, Boston, MA, USA, 2002.
- [233] Joseph M. Hellerstein, Jeffrey F. Naughton, and Avi Pfeffer. Generalized search trees for database systems. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 562–573, 1995.

VITA

VITA

Rimma V. Nehme was born in the small town of Baranovitchi, USSR (currently Belarus). She spent her childhood travelling all over the Soviet Union and East Germany because of her father's job in the military. In 1997, during her senior year in high-school, Rimma came to the United States. As a result, Rimma earned two high school diplomas in 1998, in the US and in Belarus.

Being completely fascinated with America, Rimma decided to pursue her undergraduate studies at Hillsdale College, MI in August 1998. In 2001, after three years, Rimma was granted her B.S. degree with the *Magna Cum Laude* (high honor) in Computational Mathematics. Upon graduation, Rimma joined the EMC Corporation as a software engineer in Hopkinton, MA working on the online management of enterprise storage devices. After spending several years in the industry, Rimma decided to pursue graduate studies part-time in Worcester Polytechnic Institute (WPI) in Worcester, MA and received the masters degree in Computer Science in 2005. While at WPI, Rimma shaped her research attitude and developed significant interest in database systems. Rimma then came to Purdue University to pursue her Ph.D. degree in the fall of 2005. Rimma published several papers in core database technology. In 2006, 2007 and 2008, Rimma spent three summers at Microsoft Research in Redmond, WA, one of the top research labs in the field world-wide. During her summer internships, Rimma interacted with many world-class researchers and further built her database systems experience. Rimma V. Nehme graduated with a Ph.D. degree in Computer Science from Purdue University in August 2009 and joined the Microsoft Jim Gray Systems Lab in Madison, Wisconsin USA.