Modeling and Analyzing Real-Time Data Streams

Krasimira Kapitanova, Sang H. Son
Department of Computer Science
University of Virginia
Charlottesville, VA, USA
{krasi, son}@virginia.edu

Woochul Kang, Won-Tae Kim
Electronics and Telecommunication
Research Institute (ETRI)
Daejeon, Korea
{wchkang, wtkim}@etri.re.kr

Abstract—**Achieving situation awareness is especially challenging for real-time data stream applications because they i) operate on continuous unbounded streams of data, and ii) have inherent real-time requirements. In this paper we show how formal data stream modeling and analysis can be used to better understand stream behavior, evaluate query costs, and improve application performance. We use MEDAL, a formal specification language based on Petri nets, to model the data stream queries and the Quality-of-Service (QoS) management mechanisms in a data stream system. MEDAL's ability to combine query logic and data admission control in one model allows us to design a single comprehensive model of the system. This model can be used to perform a large set of analyses to help improve the application's performance and QoS.**

Keywords-**data stream analysis; Petri nets; operator selectivity estimation; stream query modeling; QoS management**

I. INTRODUCTION

Situation awareness (SA) has been recognized as a critical foundation for successful decision-making across a broad range of complex real-time applications, including aviation and air traffic control [1], emergency response [2], and military command and control operations [3]. SA is especially important for applications where the information flow can be high and poor decisions may lead to serious consequences (e.g., piloting a plane, or treating critically injured patients). These applications need to operate on continuous unbounded streams of data to understand how incoming information and events could impact the system, both now and in the near future. The streaming data may come from various sources, such as sensor readings, router traffic traces, or telephone records. Therefore, the capability to manage data streams becomes an essential application requirement.

These applications also have inherent real-time requirements, and queries on the streaming data should be finished within their respective deadlines. Consider a surveillance system as an example. The system is expected to detect if a target enters the monitored area and alert the controlling party (e.g., human operators). If the detection does not occur within a certain deadline, the target may be missed. However, due to the dynamic nature of the input data streams, the stream queries may have unpredictable execution cost. First, the arrival rate of the data streams can be volatile, which leads to variable input volumes to the queries. Second,

the content of the data streams may vary with time, which causes the *selectivity* of the query operators to change over time. The variable stream behavior and the irregular workload pattern make maintaining the desired level of QoS a challenging task.

Formal data stream analysis can help tremendously in achieving SA. Such analysis can provide query designers with better understanding of the behavior of the real-time data streams they are working with. It could also be used to more accurately predict changes in the data arrival patterns and the query workloads. In addition, analysis of the data admission controller, responsible for determining the amount of stream data to be used as input to the queries, could aid the design of better controllers that can adapt faster to workload fluctuations. To perform such analysis, we need a specification language that will allow us to formally model real-time data streams, stream queries, as well as data admission control mechanisms. Such a language, however, is missing. To address this, we propose to use MEDAL - a formal specification language, which in its nature is an enhanced Petri net. MEDAL can capture the structural, spatial, and temporal characteristics of a complex distributed real-time system, which makes it suitable for modeling and analysis of data streams.

In this paper we describe how MEDAL can be used to model and analyze real-time data stream applications. First, each query plan can be specified with a MEDAL model. Second, MEDAL is also capable of modeling the data admission controller. MEDAL's ability to model both the queries and the data admission control enables us to combine the two main components of data stream applications (query logic and control) into a single comprehensive system model. Third, using the MEDAL query models to analyze system properties, such as the behavior of the data streams, the cost and selectivity of different query operators, and the real-time properties of the queries, can significantly improve the QoS of the applications. Further, analysis of the MEDAL model could also help admission control designers build better and more accurate prediction-based controllers.

The main contributions of this paper are: 1) Adapting MEDAL to model real-time data stream queries and stream management mechanisms; 2) Integrating query logic and

data control models into a comprehensive data stream application model; 3) Introducing how MEDAL can be used for various types of analyses, such as query plan optimization and application timeliness.

The rest of this paper is organized as follows: We discuss the related work in Section 2. Section 3 gives a short overview of MEDAL and some of its properties. Section 4 briefly introduces the components of our system model, namely, periodic query model, query plans, and the data admission controller. Section 5 describes how operator cost and selectivity predictions are used to improve the QoS of the system. Section 6 describes how MEDAL can be used to model stream queries and data admission. We demonstrate some of the types of analyses that MEDAL can be used for in Section 7, and Section 8 concludes the paper.

II. RELATED WORK

Query optimization: There have been significant research efforts devoted to the query optimization problem in distributed data stream management systems (DSMS) [4], [5]. Adaptive filters have been proposed to regulate the data stream rate while still guaranteeing adequate answer precisions [4]. Liu et al. discus the challenges for distributed DSMS and propose *dynamic adaptation* techniques to alleviate the uneven workloads in distributed environments [5]. Tu et al. use system identification and rigorous controller analysis to improve the benefits of input load shedding [6]. Wei et al. propose the idea of *just-in-time* sampling to estimate the output size of query operators and use the estimation results to control the intermediate query result propagation strategy [7].

Selectivity estimation: Selectivity estimation has been a fundamental problem in the database community since query optimizers use the estimation results to determine the most efficient query plans. Sampling [8], histograms [9], index trees [10], and discrete wavelet transforms [11] are the most widely used selectivity estimation methods. Sampling has been used extensively in traditional database management systems (DBMSs) [8], [12], [13]. It gives a more accurate estimation than parametric and curve fitting methods used in DBMSs and provides a good estimation for a wide range of data types [12]. In addition, since sampling-based approaches, as opposed to histogram-based approaches, do not require the maintenance of an auxiliary data structure, they do not incur the overhead of constantly updating and maintaining that data structure. This is especially important in the context of data streams as the input rate of the streams is constantly changing. Wei et al. were the first to consider a sampling-based approach to estimate the data stream query workloads and use the results to manage the query QoS [7]. In this paper we employ this sampling technique and use it as part of our prediction-based QoS management mechanism.

Modeling stream queries: The majority of work on data streams uses SQL or SQL-like semantics to define stream queries [14], [15]. Babcock et al. use standard SQL to model stream queries and extend the expressiveness of the language to allow the specification of sliding windows [15]. However, SQL fails to express a number of essential characteristics specific to processing data streams, such as:

i) *Data dependency and correlation among different streams in heterogeneous systems*;

ii) *Collaborative decision making*: Data stream systems are distributed, concurrent, and asynchronous. Therefore, detecting events usually requires spatial and temporal composition of ad-hoc readings;

iii) *Modeling probabilities*: Individual data readings are often unreliable which results in non-deterministic decision making. In order to tolerate such non-determinism, queries might need to be defined using a probabilistic model. SQL has no explicit support in this regard;

iv) *Graphical model*: SQL does not have a natural graphical support.

Another method for modeling stream queries was proposed for the Aurora system [16]. Aurora uses a graphical "boxes and arrows" interface for specifying data flow through the system. Compared to a declarative query language, this interface is more intuitive and gives the user more control over the exact series of steps by which the query answer is obtained. However, this approach lacks the ability to model probabilities, data dependencies and correlation, and collaborative decisions.

In this paper we use a formal specification language, called MEDAL [17], to model stream queries as well as data admission control. MEDAL combines features from Stochastic, Timed, and Colored Petri nets, which allows it to model system properties, such as collaborative decision making, temporal and spatial dependencies, heterogeneity, and non-determinism. These modeling capabilities render MEDAL very suitable for modeling stream query plans.

III. MEDAL

In this section we give a short overview of the specification language MEDAL and some of its properties. The MEDAL description of a real-time application is a 7-tuple structure $F = (P, T, A, \lambda, \beta, H, L)$, where:

1) P is the set of all places. Places can represent application state or data, and are depicted as circles in a MEDAL model. $P = S \cup E$, where S represents the data input places (T, L, and A in Figure 1), and E represents the places for higher level events (place E in Figure 1).

2) T is the set of all transitions. Transitions model various types of actions or operators and are represented by rectangle bars (transitions $T1$, $T2$, $T3$, and $T4$ in Figure 1).

3) A is the set of arcs. Arcs in MEDAL represent the flow of logic/control and are depicted as directed arrows.

4) λ is the probability/weight function for the arcs and $\lambda : A \rightarrow [0, 1]$. With λ, MEDAL adopts features from
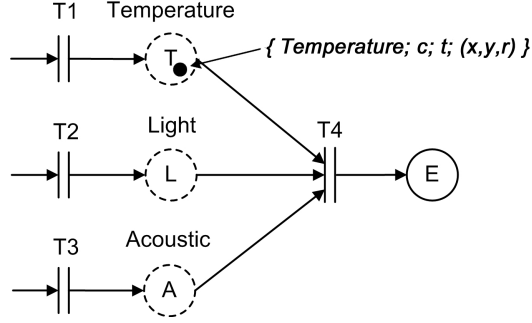
Figure 1. MEDAL model of an explosion event detection system.

Timed and Stochastic Petri nets which allows it to model probabilistic problems.

5) β is the time guard function, $\beta : T \to \cup^*(r1, r2)$, where $r1 \leq r2 \in \mathbb{R}$. For a transition T, $\beta(T) = (a1, a2) \cup (a3, a4)$ indicates that this transition can only fire during the union closure of the given ranges. β also acts as a persistency guard, i.e. we can use it to specify the amount of time a place can hold a token before this token becomes invalid.

6) H is the threshold function for places and is defined as $H : P \to \mathbb{R}$. For example, $H(p) = c$ means that a token can enter a specific place p only if its capacity is equal or higher than c.

7) L is the spatial guard function for transitions, $L : T \to \mathbb{R}^+$. It is used to guarantee that the incoming data has been generated within the radius of interest.

Figure 1 shows the MEDAL model of an example explosion detection application. An explosion is characterized by specific temperature, light, and sound values. Therefore, our detection application takes as input data streams from three types of sensors - temperature, light, and acoustic, and uses the readings to determine if an explosion has occurred. In Figure 1, the temperature, light, and sound input data streams are represented with places T, L, and A, respectively. When all three values - temperature, light, and sound, exceed some predefined thresholds, transition $T4$ fires and the application reports the detection of an explosion event, represented by place E. The stream data processing must happen in real-time since an explosion needs to be detected as soon as possible in order to evacuate people from the dangerous area.

Tokens are abstract representations of data or occurrences of events. In MEDAL, a token is defined as:

$Token = \{Type\ tp;\ Capacity\ c;\ Time\ t;\ Location\ l\}$,

where the *type* of a token indicates the type of data or event represented by the token. The *capacity* of a token is its value. It can either contain actual stream data or indicate the confidence that a particular event has occurred. Due to the high volume of data, it is often assumed that it is not possible to store a stream in its entirety, nor is it feasible to query the whole stream history. Typically, the queries are executed on a *window* of data. A window on a data stream is a segment of the data stream that is considered

for the current query. Therefore, to more accurately model the query execution behavior, rather than storing a single stream value, a token's capacity holds a whole window of data. *Time* indicates when the token was created. The *location* represents where the data carried by the token was generated. This token representation allows us to encapsulate key stream data aspects into tokens.

IV. SYSTEM MODEL

A data stream is defined as a real-time, continuous, ordered (implicitly by arrival time or explicitly by timestamps) sequence of data items [18]. A DSMS is a system especially constructed to process persistent queries on dynamic data streams. DSMSs are different from traditional DBMSs in that DBMSs expect the data to be persistent in the system and the queries to be dynamic, whereas DSMSs expect dynamic unbounded data streams and persistent queries. Emerging applications, such as sensor networks, emergency response systems, and intelligent traffic management, have brought research related to data streams in focus. These applications inherently generate data streams and DSMSs are well suited for managing the produced data.

A. Periodic Query Model

So far, DSMS research has mainly been focused on using a continuous query model [16], [19], [20], [21]. In a continuous query model, long-running continuous queries are present in the system and new data tuples trigger the query instances. These incoming data tuples are then processed and the corresponding query results are updated. The continuous model performs well when the system workload is stable and the system has sufficient resources to finish all triggered query instances. However, since the number of query instances and the system workload depend directly on the input, which could be extremely volatile, this continuous query model is not appropriate for real-time applications that need predictable responses. Another drawback of this model is that, since the query execution is driven by the data rate of the system, the application does not have control over the frequency and the deadlines of the queries. For applications where some queries are more "important" than others, it might be desirable to have the ability to execute the important queries more often. This, however, is hard to accomplish in a continuous query model.

To address this, we have developed a periodic query model (PQuery) for data stream queries with timing constraints [14]. In this periodic query model, every query has an associated period. Upon initialization, a query instance takes a snapshot of the data streams at its inputs. The query input does not change throughout the course of the query instance execution even when there are new data tuples arriving over the data streams. Instead, the newly arrived data tuples are processed by the next query instance. In this way, the query execution is not interrupted or aborted by

new incoming data. When an application receives the results of a periodic query instance, it is with the understanding that these results reflect the state of the system when the query instance was initiated. In the periodic query model, the application can specify the query periods and deadlines. These parameters can be used to calculate the number of queries in the system at any given time, which allows us to estimate the query workloads much easier than with the continuous query model.

B. Query Plan and Query Execution

A DSMS contains long-running and persistent queries. When a query arrives in the system, it is registered and is triggered periodically based on its specified period. All queries are converted to query plans (containing operators, queues, and synopses) statically before execution. Queues in a query plan store the incoming data streams and the intermediate results between the operators. A synopsis is associated with a specific operator in a query plan, and stores the accessary data structures needed for the evaluation of the operator. For example, a JOIN operator may have a synopsis that contains a HASH JOIN index for each of its inputs. When the JOIN operator is executed, these hash indices are probed to generate the JOIN results.

Consider the following example scenario where a traffic monitoring system has been deployed to constantly analyze the traffic in a particular city and determine the most suitable times for delivering supplies to the grocery stores. To achieve the desired situation awareness, the system analyzes data streams from speed and traffic sensors. We perceive events as the fundamental building blocks of a situation. Therefore, achieving situation awareness requires that we identify a particular set of events and perform our situation assessment and analysis based on their occurrence. One of the events that our traffic monitoring application is interested in is trucks that travel during light traffic in specific lanes, such as non-HOV lanes. For its consequent traffic analysis, the application calculates the average speed of these trucks. The SQL data stream and query specifications are given as follows:

Stream : $Speed$ ($int\ lane$, $float\ value$, $char[8]\ type$);
Stream : $Traffic$ ($int\ lane$, $char[10]\ type$);
Relation : $Lanes$ ($int\ ID$, $char[10]\ type$, $char[20]\ road$);

Query : $SELECT\ avg\ (Speed.value)$
    $FROM\ Speed\ [range\ 10\ minutes],\ Lanes,$
        $Traffic\ [range\ 10\ minutes]$
    $WHERE\ Speed.lane = Lanes.ID$
        $AND\ Lanes.ID = Traffic.lane$
        $AND\ Speed.type = Truck$
        $AND\ Traffic.type = Light$
    $Period\ 10\ seconds$
    $Deadline\ 5\ seconds$

The query above operates on data streams generated by speed and traffic sensors and calculates the average speed
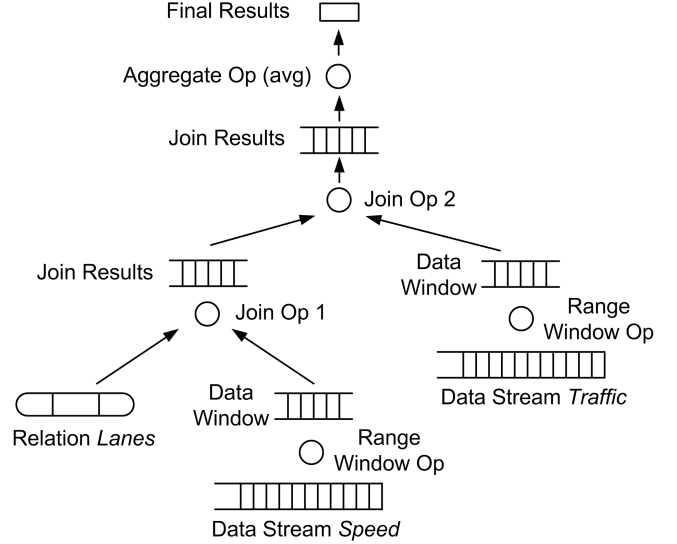

Figure 2. An example query plan.

of trucks in particular lanes during light traffic over a 10-minute window. The query needs to be executed every 10 seconds and the deadline is 5 seconds after the release time of every periodic query instance. The query plan generated based on this query is shown in Figure 2. It contains three types of query operators (RANGE WINDOW operator, JOIN operator and AGGREGATE operator) and two types of queues (one for storing the output of the RANGE WINDOW operators and one for storing the output of the JOIN operators).

After the query plan is generated, the operators are sent to the scheduler to be executed. Depending on the query model (e.g., continuous or periodic), a scheduling algorithm, such as round-robin or earliest deadline first, could be chosen so that the system requirements are met.

C. Data Admission Controller

In many real-time applications, partial results are more desirable than queries missing their deadlines. Therefore, the system might trade off data completeness for better query miss ratios at run time. We have designed an overload protection mechanism called *data admission controller*, which trades data completeness for better query miss ratios [14]. The basic approach is to reduce the incoming data volume when the system becomes overloaded. The load shedding process is performed before the data stream tuples are processed by the queries. Since it is possible that certain data stream tuples are more important than others, the data stream sources can mark their important data tuples with high importance flags. All data tuples with flags are admitted by the system.

The data admission process is controlled with a proportional-integral (PI) controller as it is simple to use and provides acceptable response time to workload fluctuations.
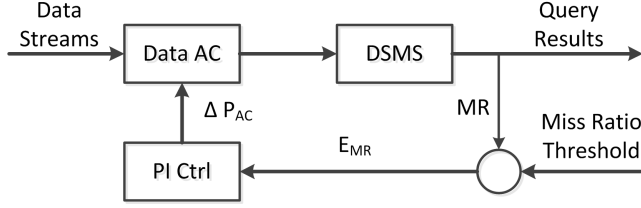
Figure 3. Data admission controller.

Operator	Average Cost (μs)
SELECTION	0.16 - 0.3
PROJECTION	0.16 - 0.2
JOIN	3 - 6
STREAM JOIN	3 - 100
DISTINCT	0.16 - 0.3
EXCEPT	0.16 - 0.3
GROUP AGGREGATE	0.16 - 0.6

Table I
OPERATOR COST

A proportional-integral-derivative (PID) controller is not suitable in this situation because of the dramatic changes that might occur in the workloads of query systems from one sampling period to another. Adding a derivative control signal amplifies the random fluctuations in the system workloads [22]. The data admission control architecture is shown in Figure 3. The query miss ratio (MR) is sampled periodically and compared against the miss ratio threshold specified by the application. The result is used by the PI controller to calculate the data admission control signal ΔP_{AC}. The control signal is derived with the equation:

$$\Delta P_{AC} = P_{MR} \times (MR_{ST} - MR_t) + I_{MR} \times (MR_{LT} - MR_t),$$

where MR_{ST} and MR_{LT} are the short-term and long-term query miss ratios sampled in the last sampling period. MR_t is the maximum miss ratio allowed by the application. P_{MR} and I_{MR} are controller parameters that specify the significance of the short-term and the long-term query miss ratios when calculating the data admission control signal. The controller parameters determine the behavior of the controller. The process of tuning these parameters, i.e. *controller tuning*, is not the focus of this paper. Readers are referred to [23] and [24] for details on the analysis and tuning of controllers.

V. PREDICTION BASED QoS MANAGEMENT

When a system becomes overloaded, more queries start missing their deadlines. If the query miss ratio increases above the application threshold, the application can no longer satisfy its QoS requirements. When this happens, in order to decrease the deadline miss ratio, the data admission controller should adjust the amount of input data sent to the queries. To perform a correct adjustment, the data admission controller needs information about the future workloads and the corresponding query execution times. We predict the query workload using execution time *profiling* and input data *sampling* [7]. To estimate the query execution time, we need three parameters for each query, namely, the input data stream volume, the *operator selectivity*, and the *execution overhead* per data tuple for each operator. We make the assumption that the input data volumes for queries that are ready to execute are known. In the rest of this section we describe in more detail the prediction techniques employed by our QoS management mechanism.

A. Operator selectivity estimation

Operator selectivity: The operator selectivity measures the ratio between the size of the input that is passed to an operator and the size of output data remaining after the operator has been applied. The selectivity of an operator is defined as: $Selectivity = Size_{output}/Size_{input}$

As the operator selectivity varies, the size of the output changes even when the input volume stays the same. Thus, the query execution cost could change significantly when the data stream content changes. While possible to design the system based on the maximum system workloads, this is not practical nor efficient, because the transient system overload does not persist for long periods of time. Therefore, we dynamically estimate and update the operator selectivity based on changes in the workload.

Selectivity estimation using sampling: We use sampling as selectivity estimation algorithm because it is well-studied, easy to implement, and yields good estimation when handling high-rate data streams [12]. A *sampler* query plan is constructed for every query in the system. These sampler query plans are the same as their corresponding real query plans. When a query instance is released to the scheduler, the sampler is executed first with sampled data tuples from the input. The data tuples are randomly selected from the real input according to a predefined sample ratio. The results of the sampler queries are used to estimate the selectivity and execution time of the operators in the real query plans.

B. Operator Overhead Prediction

Operator Cost Estimation: We make the assumption that all data, including incoming data streams and intermediate results, is stored in memory. Therefore, we can estimate the time it takes for an operator to process a data tuple, excluding any additional delays caused by fetching data from the disk. Table I shows the average execution time per tuple for different types of operators. We can see that, with the exception of JOIN operators, the execution cost of most operators is relatively small and fairly predictable. The cost of a STREAM JOIN (a JOIN operator on two streams) varies from $3\mu s$ to $100\mu s$ because the number of data tuples selected from the data streams could vary significantly. For more details on how the cost analysis of these operators is performed, the reader is referred to our previous work [7].
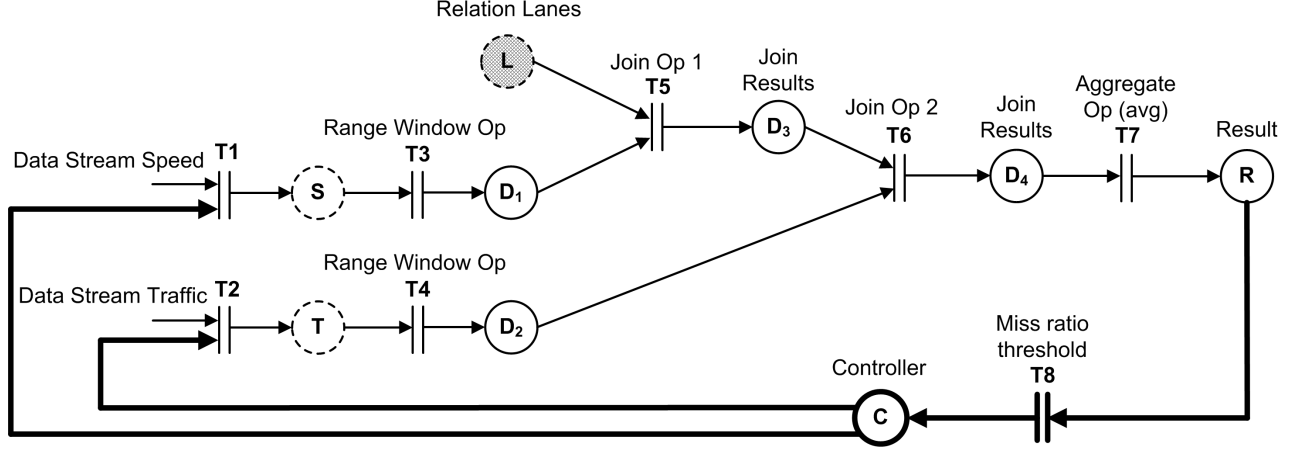
Figure 4.   MEDAL query plan.

Maintaining Cost Constants Using Profiling: There are a number of constants that participate in the cost analysis of query operators. For example, when estimating the cost of a SELECTION operator, one of the necessary inputs is the execution time to evaluate the predicates. The system needs to know the precise values of such cost parameters in order to accurately estimate the execution time of the operators. The approach we use is to keep track of the execution time of each operator and compute these cost parameters periodically. Suppose that after a query instance, the execution time for an operators is T_{exec}. The new cost parameter C_{new} is calculated based on the old cost parameter C_{old} and T_{exec}. We use the single exponential smoothing formula:

$$C_{new} = C_{old} \times (1 - \alpha) + T_{exec} \times \alpha,$$

where α is the smoothing factor and $0 < \alpha < 1$. In order to give relatively higher weights to recent observations, we use values of α close to 1, which causes a smaller smoothing effect and gives greater weight to recent changes in the data.

VI. MODELING QUERIES AND CONTROL

An advantage of MEDAL is that it can be used to specify a more comprehensive application model by combining the admission controller and the query plan models into a single MEDAL model. This is very beneficial for two main reasons. First, it allows us to model the direct relationship between the data admission controller and the query inputs. Second, integrating the query plan models and the admission controller logic gives us a better understanding of the correlation between the query results and the stream input volume.

A. MEDAL query plans

MEDAL can seamlessly be applied to modeling real-time query plans. The same query plan from Figure 2 is specified in Figure 4 using MEDAL. The different types of query operators are modeled with the help of transitions (transitions

T1 - T7). Application state and data are represented using places. Places with interrupted border are used to model data input (places *S*, *T*, and *L*). We introduce the use of shaded places (place *L*) in order to distinguish the input relations, which tend to be static for the most part, from the input data streams. Therefore, the set of places *P* in MEDAL can now be expressed as $P = S \cup R \cup E$, where *S* represents the input streams, *R* represents the input relations, and *E* represents the higher level events and intermediate results.

B. Data admission control

There is an array of real-time data stream applications that use feedback control to adjust to the constantly changing environment and improve system performance [25], [26]. To model feedback control, MEDAL employs a control theoretic approach which adopts the controller synthesis paradigm from control theory. Given a model of the system's dynamics and a specification of the desired closed-loop behavior, the objective is to synthesize a controller to achieve the specified behavior. This approach relies on the clear distinction between the system and the controller and requires that the information flow between them is explicitly modeled. We meet this requirement by designing two separate MEDAL models - one for the query plan and one for the data admission controller. The two models are then composed into a single application MEDAL model which allows us to analyze the interactions between the system and the data admission controller.

The feedback control loop in Figure 4 is shown in bold. Transition *T8* compares the query deadline miss ratio to a predefined miss ratio threshold. If the observed miss ratio is higher, controller *C* calculates the new data admission ratio and propagates it to the system. Place *C* is a very high-level representation of a controller. A more detailed model of the controller's logic can be designed with MEDAL and used to replace place *C*. Due to space limitations, we do not discuss the details of modeling the internal controller logic.

MEDAL can be used to model different levels of abstraction. For example, analogously to using place *C* to model the entire controller logic, we can abstract away the query plan from Figure 4. The query plan could be modeled with a single place, similarly to how Figure 3 models a DSMS. This is especially helpful during the controller design phase, since it allows us to treat the query as a simple linear system without considering how the internal parts of this system interact. Using this simplified model of the query plan, the appropriate values for the P_{MR} and I_{MR} controller parameters are determined through system identification [27]. Once the controller is designed, we can expand the query plan model and use it to additionally adjust the controller to better reflect the query properties. The same abstraction approach can also be applied when there are multiple registered queries. The whole query set is abstracted away using a single place, which can consequently be expanded once the controller is designed.

Another requirement introduced by the control theoretic approach is that changes in the system are observable [28]. MEDAL achieves *observability* by explicitly identifying and modeling the parameters that need to be monitored. In Figure 4, the query results are delivered to controller *C* through transition *T8*. As mentioned above, this transition determines the difference between the query deadline miss ratio and the threshold specified by the application. Therefore, transition *T8* allows us to explicitly model the query miss ratio and thus achieve the required *observability*.

VII. MEDAL ANALYSIS

A. *Query optimization analysis*

The MEDAL model of a data stream system could be used to analyze the query plans, the controller logic, as well as the interactions between them. This analysis could help designers identify possible query plan optimizations and design more suitable data admission mechanisms. Consider, for example, the following scenario: For the query model on Figure 4 we have estimated the cost of each of the query operators. In addition, based on historical workload information, we have also estimated the average selectivity of the operators. The data admission controller has been designed to decrease the data admission rate by x%, where *x* is a function of the difference between the miss ratio threshold and the current deadline miss ratio. We can use the MEDAL model of this application to determine if the controller logic has the desired effect on the deadline miss ratio. Using the available historical data as input to the MEDAL model, we can estimate the new deadline miss ratio, and thus evaluate the controller's efficiency. This is a very simple high-level example, but it showcases one of the possible MEDAL system analyses.
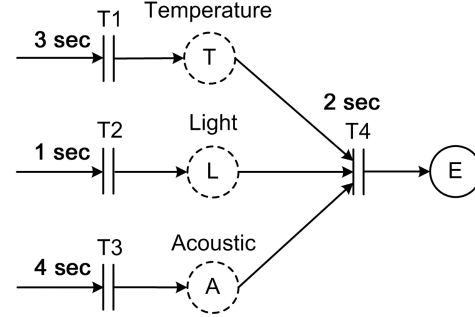

Figure 5. Using MEDAL for real-time analysis.

B. *Real-time stream analysis*

MEDAL can also be used for real-time analysis of data steam network applications. For example, a safety requirement for the explosion detection application could be that explosions are detected within 5 seconds of their occurrence. If the application takes longer to detect an explosion and send an alert, there might not be enough time to perform evacuation. The MEDAL model of the explosion application could be used to help determine if the application is able to meet this requirement. During the design of the application, the sampling periods for the temperature, light, and acoustic sensors have been set to 3, 2, and 4 seconds, respectively. In addition, based on knowledge about the system and the environment, the designers have determined that it takes 2 seconds, in the worst case, to process the data, run the explosion detection algorithms, and calculate the result. Since the rest of the transitions in the model represent simple computations, the execution times associated with them could be neglected for this particular application analysis.

Figure 5 shows the time information associated with each transition. Analyzing this model reveals that detecting an explosion could take 6 seconds in the worst case: 4 seconds for sound detection and 2 more seconds for data processing. This, however, means that the application may not be able to meet its real-time requirements. Therefore, either the acoustic sensor sampling period has to be changed, or the system should be altered such that analyzing the incoming data does not take more than 1 second. Similar real-time analysis could be performed for much more complex systems.

VIII. CONCLUSIONS AND FUTURE WORK

In this paper we show how MEDAL can be used to model and analyze real-time data stream queries, QoS management mechanisms, and the relationships between them. Unlike previous work, where query models and system control logic were designed and analyzed separately, MEDAL allows us to merge these two components into a single more comprehensive system model. The advantages of this combined model is that it can be used not only to predict the workload and estimate the query cost, but also to model and analyze the interactions between the input and output of the query

plans and the data control mechanism, which gives us a much better understanding of the system.

For future work we plan to design an automated data admission controller that can preemptively alter the data admission rates and thus maintain high data completeness while ensuring low deadline miss ratio. This data admission controller will take advantage of context information, such as spatial and temporal data characteristics, as well as stream behavior patterns extracted from historical data. We will also implement a MEDAL tool, which will allow system designers to conveniently build and analyze data stream applications. This tool could be used to perform operator cost analysis and selectivity estimation. It will also help model dependencies between the data stream system and the surrounding environment, which could be extremely useful for context-aware workload prediction.

REFERENCES

[1] R. Nullmeyer, D. Stella, G. Montijo, and S. Harden, "Human factors in air force flight mishaps: Implications for change," *I/ITSEC*, 2005.

[2] A. Blandford and B. L. W. Wong, "Situation awareness in emergency medical dispatch," *International Journal of Human-Computer Studies*, vol. 61, pp. 421–452, 2004.

[3] J. C. Gorman, N. J. Cooke, and J. L. Winner, "Measuring team situation awareness in decentralized command and control environments," *Ergonomics*, vol. 49, pp. 1312–1325, 2006.

[4] C. Olston, J. Jiang, and J. Widom, "Adaptive filters for continuous queries over distributed data streams," in *SIGMOD*, 2003.

[5] B. Liu, Y. Zhu, M. Jbantova, B. Momberger, and E. Rundensteiner, "A dynamically adaptive distributed system for processing complex continuous queries," in *VLDB*, 2005.

[6] Y.-C. Tu, S. Liu, S. Prabhakar, and B. Yao, "Load shedding in stream databases: a control-based approach," in *VLDB*, 2006.

[7] Y. Wei, V. Prasad, S. Son, and J. Stankovic, "Prediction-based QoS management for real-time data streams," in *RTSS*, 2006.

[8] P. Haas, J. Naughton, and A. Swami, "On the relative cost of sampling for Join selectivity estimation," in *PODS*, 1994.

[9] V. Poosala and Y. Ioannidis, "Selectivity estimation without the attribute value independence assumption," in *VLDB*, 1997.

[10] D. Comer, "The ubiquitous B-Tree," *ACM Computing Surveys*, vol. 11, pp. 121–137, June 1979.

[11] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery, *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, 1992.

[12] D. Barbara, W. DuMouchel, C. Faloutsos, P. Hass, J. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, K. Ross, and K. Sevcik, "The New Jersey data reduction report," Bulletin of the Technical Committee on Data Engineering, Tech. Rep., 1997.

[13] S. Chaudhuri, G. Das, M. Datar, R. Motwani, and V. Narasayya, "Overcoming limitations of sampling for aggregation queries," in *ICDE*, 2001.

[14] Y. Wei, S. Son, and J. Stankovic, "RTSTREAM: Real-time query processing for data streams," in *ISORC*, 2006.

[15] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom, "Models and issues in data stream systems," in *PODS*, 2002.

[16] D. Abadi, D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. Zdonik, "Aurora: a new model and architecture for data stream management," *The VLDB Journal*, vol. 12, pp. 120–139, 2003.

[17] K. Kapitanova and S. Son, "MEDAL: A compact event description and analysis language for wireless sensor networks," *INSS*, 2009.

[18] L. Golab and M. T. Özsu, "Issues in data stream management," *ACM SIGMOD Record*, vol. 32, pp. 5–14, 2003.

[19] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tabul, and S. Zdonik, "Monitoring streams - a new class of data management applications," in *VLDB*, 2002.

[20] D. Abadi, Y. Ahmad, M. Balazinska, U. Çetintemel, M. Cherniack, J. Hwang, W. Lindner, A. Maskey, A. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the Borealis stream processing engine," in *CIDR*, 2005.

[21] L. Girod, K. Jamieson, Y. Mei, R. Newton, S. Rost, A. Thiagarajan, H. Balakrishnan, and S. Madden, "WaveScope: a signal-oriented data stream management system," in *SenSys*, 2006.

[22] J. Hellerstein, Y. Diao, S. Parekh, and D. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[23] C. Lu, Y. Lu, T. Abdelzaher, J. Stankovic, and S. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Trans. Parallel and Distributed Systems*, vol. 17, pp. 1014–1027, 2006.

[24] C. Lu, J. Stankovic, S. Son, and G. Tao, "Feedback control real-time scheduling: Framework, modeling, and algorithms," *Real-Time Systems*, 2002.

[25] S. Lin, J. Zhang, G. Zhou, L. Gu, J. Stankovic, and T. He, "ATPC: adaptive transmission power control for wireless sensor networks," in *SenSys*, 2006.

[26] T. He, J. Stankovic, C. Lu, and T. Abdelzaher, "SPEED: A stateless protocol for real-time communication in sensor networks," in *ICDCS*, 2003.

[27] L. Ljung, *System identification (2nd ed.): theory for the user*. Prentice Hall PTR, 1999.

[28] E. Lee and L. Markus, *Foundations of optimal control theory*, 1967.