Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 04-029

Probabilistic Stream Relational Algebra: A Data Model for Sensor Data Streams

Haiyang Liu, San-yih Hwang, and Jaideep Srivastava

July 12, 2004

Probabilistic Stream Relational Algebra: A Data Model for Sensor Data Streams

Haiyang Liu[a], San-Yih Hwang[b], and Jaideep Srivastava[a]
[a]Department of Computer Science and Engineering , University of Minnesota
[b]Department of Information Management, National Sun Yat-Sen University

Abstract

Sensor data streams exhibit special characteristics such as inherent information uncertainty and inherent data sample correlations, both within and across streams. We introduce a new data model, called Probabilistic Stream Relational Algebra (PSRA), that models a sensor data stream as a set of probabilistic data samples, along with prediction strategies for each attributes, capturing domain knowledge of inherent data correlations. We also explicitly associate every operation with schedule, specifying when next data sample should be produced, to facilitate resource management in sensor networks. We prove that operators in PSRA are non-blocking, thus making PSRA especially suitable for data stream processing. We also show that conventional relational model and existing deterministic data stream processing model can be modeled in PSRA.

1. Introduction

Advances in device miniaturization, wireless networking and embedded processing have reduced both the size and cost required for sensing, communicating and computing. This promises a future where a large number of interconnected and possibly collaborative sensors will be deployed in many applications such as environmental monitoring, industrial sensing and diagnostics, battlefield monitoring and patient monitoring [BGS00]. Large amounts of data, in the form of streams are produced in such environments. These data streams, referred as sensor data streams, exhibit several key characteristics in addition to those outlined in [BBD+02]:

Inherent Uncertainty

Sensor data streams generated from sensor readings are discrete observations of generally continuous physical phenomena. The data samples in a sensor data stream depict only a partial picture of the phenomena under observation. Thus, sensor data streams are inherently approximate representations of physical phenomena. Uncertainty is therefore inherent when applications try to query the physical world with only discrete sensor data streams available. In addition, uncertainties are also introduced during the processes of data collection (i.e. sensor measurement), data movement (i.e. communication) and data processing (i.e. computation). These uncertainty sources are summarized as follows:

Data Collection: Data regarding a phenomenon are collected through sensor measurements. Measurement errors are inherent due to the limitations of sensing principles and sensor operation environments, such as stability of power sources, sensor locations and sampling frequencies. Physical measurements usually come as a value distribution range, generally modeled as a Gaussian probability distribution function (p.d.f.).

Data Movement: In sensor networks, data are moved through networks of communication channels. Data delay and data loss due to limited communication resources such as channel bandwidth are inherent in today's communication networks [CC+03]. Even if a perfect measurement at a required time point is available at the source side (i.e. the sensor), data delay and data loss can introduce uncertainty at the sink side (i.e. end user side) because the data sample may not arrive on time or never arrive.

Data Processing: Since sensor data streams are potentially unbounded in size, data processing units with only limited storages usually have to use approximate methods such as sliding windows or data stream sketching to process data [BBD+02]. Furthermore, stream data processing often has real-time requirements, with only a milited time available to process each sample. Thus, the limitation of computing resources can also introduce uncertainties in the results.

Inherent Intra- and inter- stream correlations

Data samples in a sensor data stream are usually temporal observations of physical phenomena. Thus, these data samples have inherent temporal correlations as possessed by the phenomena. For example, vehicle positions at time t_0 and t_1 are correlated through the vehicle velocity and the time difference (t_1- t_0). Different data streams observing the same object from different aspects can be correlated as well. For example, the temperature reading stream and the pressure reading stream of an air chamber are correlated through the ideal gas law: $PV = nRT$, where P denotes pressure, V volume, T temperature, R a constant and n related to the mass of the air in the chamber. When the chamber is closed and volume is fixed, the correlation is simplified to P/T = constant.

Energy consumption sensitive

The small size of sensors and highly demanding functions in sensing, communication and computing make energy a precious resource in sensor data stream processing systems. When sensors are networked through wireless channels, communication is the major energy consumer [EMS02]. Thus, when and how frequently data samples are transmitted plays a major role in prolonging the lifetime of a sensor network.

Data importance is context-dependent
In a sensor data stream, data samples have different importance under different computing context (i.e. everything that could affect performing an operation). For example, if the computing context is evaluation of a predicate (Temperature < 30), temperature readings with values around 30 will be more important than those far below 30 or far above 30 because the readings around 30 contribute more to the determination of state change. This example illustrates data value-based importance. Data importance also depends on other attributes of data streams. For example, if the computing context is to obtain the most accurate temperature of an object. Assume two temperature sensors are mounted on the same location of an object. The temperature readings from the temperature sensor with higher precision have more importance than those from the sensor with lower precision. Data importance plays a central role in load shedding when a data steam processing system is overloaded.

Efficient and systematic management of sensor data streams calls for a data model that takes into account the above special characteristics. In this paper we develop the Probabilistic Stream Relational Algebra (PSRA), a data model for sensor data steams, by extending the conventional relational model. We first extend the relational model to allow new data types supporting data uncertainty representations, e.g. example Gaussian p.d.f. Then domain knowledge of applications, capturing the intra- or inter- stream correlations is incorporated into the model. Thus, a sensor data stream is modeled as a series of data samples and the knowledge describing their correlations. The correlation model and data samples with support for uncertainties make up a more complete view of a phenomena being monitored. This new data stream model addresses the first two issues outlined above. In order to address the last two issues, every sensor data stream operation is associated with a schedule, specifying when to produce the next data sample. A predicate-based schedule specification is proposed, supporting specification of both push (e.g. event-driven) and pull (e.g. periodic query) type operations. Finally, we prove that both conventional relational model and deterministic data stream processing model are special cases of PSRA. We also illustrate the power of the model in helping resource management in sensor networks through clear formulation of the resource optimization problems for the best effort and the QoS-driven data stream processing modes.

This paper is organized as follows. Section 2 introduces new concepts such as Predication Strategy to be used in PSRA and how basic operations (e.g. algebraic operations and predicate evaluation) are carried out on new data types supporting uncertainties. Section 3 defines the concept of PS-relation, the formal model of sensor data streams. The essential relational operators (i.e. Union, Intersection, Difference, Select, Projection, Cartesian Product, Join, Aggregation) over the PS-relation are defined in section 4. Several important properties of PSRA are discussed in section 5. Related work is discussed in section 6. We conclude this paper in Section 7.

2. Preliminaries and Basic Definitions

2.1 Representation of data uncertainty

Data uncertainty management generally falls into two categories: fuzzy theory-based and probability theory-based. Since the nature of data management in sensor network applications is fundamentally probabilistic [FGB02], we adopt a probability theory-based approach. In probability theory, uncertainty is generally represented using a Probability Density Function (p.d.f.), which can be either discrete or continuous [R95].

Continuous P.D.F..
Representative continuous p.d.f.'s include exponential distribution, Gaussian distribution etc [R95]. Gaussian p.d.f is of particular interest to sensor network applications because measurements of physical phenomena are generally modeled as Gaussian distributions [W94]. The Central Limit theorem reveals that a large number of random samples leads to a Gaussian distribution. A Gaussian pd.f. (also known as Normal p.d.f.), is characterized by two parameters: μ – the mean values and σ – the standard deviation. Formally,

$$N(\mu,\sigma) = \frac{e^{-(x-\mu)^2/2\sigma^2}}{\sqrt{2\pi}\sigma}$$

Discrete P.D.F.
A Discrete p.d.f. is given by specifying the probability of each value a random variable could assume. Well known discrete p.d.f.'s include uniform distribution, binomial distribution, geometric distribution, and Poisson distribution [R95]. The discrete p.d.f. for a Boolean random variable is a special one whose domain contains only two values: TRUE and FALSE.

2.2 Essential strategies for sensor data stream processing

srediction strategy
In sensor network applications, data streams are observations of physical phenomena. Data samples in sensor data streams arising either directly from sensor measurements (referred to as *original streams*) or derived (through operations) from original streams (referred to as *derived streams*) have intrinsic correlations. For example, a stream of temperature readings of a room must follow heat transfer laws, e.g. the room temperature change is proportional to the room heat capacity change, which in turn is determined by the energy the room received in a certain period of time [M95]. Thus, estimation of the temperature change in a time unit is bounded by the

maximum energy the room could receive. Furthermore, if the amount of energy received by the room is known, the next temperature reading can be predicted (with a certain level of uncertainty). We refer to the correlation between data samples within a stream as *intra-correlation*. In some cases, data samples in stream A are correlated with data samples in stream B. For example, the temperature and pressure of a closed chamber filled with air (generally treated as idea gas) are governed by the ideal gas law. Thus, at any time point since the temperature measurement can be predicted through the pressure measurement at the same time, and *vice versa*, we refer to this correlation as *inter-correlation.* Taking into account these correlations can significantly reduce number of data sample required to carry same amount of information. Thus modeling a sensor data stream as an ordered collection of data samples with timestamps is oversimplified. These intrinsic data correlations associated with a stream must be captured to model the stream as an integral entity. In the PSRA model, these data correlations are modeled as Prediction Strategies, formally defined as follows.

Definition 1: A *stream history* is a sequence of ordered pairs (t_i, P_i) listed in ascending order of t_i, where P_i is the data sample value at time t_i.

Definition 2: *Prediction Strategy* (*P-strategy*)

A *P-strategy* is a function $\mathcal{P}$: (*history*, *t*) $\rightarrow P$ that outputs (or predicts) a value in *P* given a sequence history *history* and an arbitrary timestamp *t*.

A prediction strategy is an approximation model for an attribute value in a sensor data stream in case sensor reading is not available or the sample was lost in communication when the attribute value at a certain time point is needed. Note that the range (*P*) of a *P-strategy* does not have to be the same as any domain in the input history (*history*). The above pressure-temperature inter-correlation example is a case of using attributes in other domains, possibly in other streams, for prediction. We also do NOT constrain the range of *t* to allow backward predictions (i.e., *t* does not have to be larger than any timestamp in *history*). Two trivial but important strategies CONST and IGNORANT are listed in Appendix A.1. CONST P-strategy models a constant data stream where attribute values do not change over time. Obviously, only a single tuple at any time instant is needed to perfectly represent the whole constant data stream. Attributes in the traditional relational model can be readily modeled with this strategy. IGNORANT P-Strategy models the situation where no data correlation model is available. Thus if there is a measurement at time instant t, the prediction for that time instant is exactly the same measurement. Otherwise, IGNORANT P-Strategy returns the most recent available measurement value with data uncertainty being infinite, which renders the result completely unreliable. Data behaviors in deterministic stream models can be readily modeled with IGNORANT P-Strategy since there is no prediction available and estimation of the states of observables only comes from timely availability of data.

In practice, prediction strategies for sensor data streams can be constructed from physical model or statistical model of observations. The ideal gas law in the above example would yield a physical-law-based prediction strategy for the temperature stream. For general linear system models, Kalman Filter [K60] is an excellent tool to build an optimal observer combining non-perfect knowledge of system model and non perfect measurements. Kalman filter has been extensively used in many physical measurement related engineering applications such as aircraft tracking and navigation [BL98]. For example, in an aircraft tracking application, to determine the velocity of an aircraft, one could use a Doppler radar, or the velocity indications of an inertial navigation system, or the pitot and tatic pressure and relative wind information in the air data system. A Kalman filter can be built to combine all of this data and knowledge of the various systems' dynamics to generate an overall best estimate of velocity at any time [M79]. A particle position tracking sample problem is given in [K60]. Details are omitted here. When physical models are not available, statistical models such as those data stream sketching models can be used to build the prediction strategies. Examples include randomized sketching [AMS96], V-Optimal Histograms [JK+98] and wavelet-based approximation [CG+00]

P-strategies for derived PS-relations can be constructed out of input ones based on the operations. How P-strategies of sensor data streams evolve through each PS operator is discussed in Section 4. When P-strategies constructed this way do not generate good predictions, new ones can be constructed through directly building statistical models.

Composition strategies for probabilistic data

The introduction of data uncertainty into the sensor data stream processing model makes operations much more complicated as all operations now need to handle p.d.f.'s instead of deterministic values. This is especially true when it comes to combining multiple data sample into a new one. For example, observations of the same object at the same time can be obtained from multiple sources such as redundant sensors. The final estimation of the state of the object may be specified as the average of the individual observations. Generally operations involving multiple random variables need to have the joint distribution function of those variables. Constructing the joint distribution from individual p.d.f.'s needs to consider the dependency of these random variables. In our framework, a dependency and requirement model, referred as Composition Strategy, is introduced to define the way multiple random variables are "composed" through an operation. This model allows users to incorporate their domain knowledge (i.e. the extent to which these variables are dependent) and their requirements (e.g. conservative or aggressive) to a data stream processing system.

The dependency and requirement model specifies the

dependencies of multiple random variables and provides the guidance on how an operation involving multiple random variables should be carried out. Since Gaussian distribution is the most commonly used p.d.f. for measurement representation, we use it to show some example results in the dependency and requirement model. Table 1 and 2 in Appendix A.2 show the composition strategies for algebraic operations and logical operations respectively.

Cleaning Strategy

In a sensor data stream processing system, observations of the same observable at same time can be obtained from multiple sources and paths. In order to form a single consistent view of the observable at a given time, a user-specified strategy is needed to fusion the multiple observations available (either consistent or inconsistent) of the same attributes at the same time point into one estimation for that time point. A cleaning strategy is introduced for this purpose.

Definition 3: A stream observation is an ordered pair (t, P_s), where P_s is a set of observed values at time point t.

Definition 4: *Cleaning Strategy* (*CL-strategy*)

A CL-strategy is a function C: *obs* $\rightarrow P$ that outputs a value in P by combining values in a stream observation *obs*.

A cleaning strategy specifies the way to generate a single estimation of an observable at any time point, eliminating redundancies and inconsistencies inherent in a probabilistic sensor data stream processing system. Common cleaning strategies include:

1. Optimistic -- picking the observation with the least uncertainty
2. Average – taking the average of all observations
3. Conservative -- picking the observation with the most uncertainty.

A cleaning strategy can be composed using the basic operations with appropriate settings on dependency and requirements as described above. For example, suppose a stream observation contains three positively correlated values $P_s = \{N(\mu_1, \sigma_1), N(\mu_2, \sigma_2), N(\mu_3, \sigma_3)\}$, and the average operation is adopted as the CL-strategy. By taking the aggressive requirement, the resultant value becomes

$N(\frac{\mu_1+\mu_2+\mu_3}{3}, \frac{\min(\sigma_1,\sigma_2,\sigma_3)}{3})$ (referred to Table 1 in Appendix A.2.).

2.3 Schedules

Though the states of many physical phenomena are continuous, their representations and processing must be carried out on discrete time points. We introduce the concept of schedule to specify these time points required by applications, triggered by special events or limited by sensor capabilities.

Definition 5: ***Schedule***

A *schedule* S is a list of (finite or infinite) monotonically increasing timestamps.

A schedule defines the series of all distinct time points that are of interest to an application or an operation, in the entire time history. In many sensor network applications, a schedule consists of an infinite number of time points. For example, "*starting from 12:00:00 05/15/04 with a sampling frequency of 10 minutes*" specifies a periodic schedule commonly used in monitoring applications.

To construct a practical sensor network application, a finite representation of a schedule is needed. Observe that, though a schedule can have an infinite number of members, only a finite number of them are needed at any time. In most cases, only the next time point in a schedule is needed for a sensor or processing node to operate. To support both push-type and pull-type applications, the schedule specification must be able to support "event-driven" specification and application requirement-driven such as periodic monitoring requirements. We propose a predicate-based schedule specification for this purpose, which is shown in Appendix A.3.

3. Probabilistic Stream Relation (PS-relation)

3.1 Probabilistic Stream Tuple

Borrowing the concepts from OLAP [CD97], we distinguish data in sensor network applications into two categories, dimension data and measurement data. Dimension data specifies a category of information for identifying objects rather than measuring readings. Dimension data are commonly used to identify data sources and take deterministic forms (i.e. represented in traditional data types instead of p.d.f.'s). Sensor id's, sensor types, and sensor locations are among the examples of dimension data. Measurement data specifies the attributes of the object identified by the associated dimension data. Temperature readings are examples of measurement data in a data stream generated by a temperature sensor. As discussed in previous sections, measurement data have inherent uncertainties and thus are represented generally by P.D.F.s. Furthermore, domain knowledge and application requirements are introduced in form of prediction strategy to capture the temporal correlations of measurement data. Thus there is a need to extend the traditional definition of data type defining an attribute. We introduce the concept of stream domain to capture the extra characteristics.

Definition 6: ***Probabilistic Stream Domain (PSD)***

A probability stream domain (PSD) is an ordered pair (*pdd*, *p-strategy*), where *pdd* is referred to as Probabilistic Data Domain (PDD) whose values may be p.d.f.'s in addition to deterministic data types, and *p-strategy* is a prediction strategy that generate values in the domain of *pdd*.

The *pdd* in a PSD specifies both the data type (e.g., Gaussian p.d.f.) and the data range (e.g. from 0 to 1000), which can be used to specify the limitation of a sensor. The *p-strategy* in a

PSD captures the stream characteristic. Note that *p-strategy* is extra information for facilitating modeling of a data stream, rather than a mandatory requirement on the data. In fact the *p-strategy* will be dynamically updated as additional observations in a data stream come in.

Definition 7: Dimensional Domain (DD)

A DD domain is a special case of a PSD domain, where *pdd* has a deterministic data type and *p-strategy* is CONST

A DD domain provides an approach to modeling conventional data domain in our probabilistic data stream framework. An attribute in a DD domain is always deterministic and the CONST prediction strategy guarantees that its value does not change over time. Traditional non-stream relational data are readily modeled using DD domain. Note that traditional deterministic data type can be modeled as special cases of probabilistic data type for the purpose of computation. For example, deterministic value 100 is equivalently represented by a Gaussian p.d.f. N(100, 0), i.e. a Gaussian distribution function with mean value at 100 and standard deviation being 0. This special function is also referred to as Delta function [R95].

Definition 8: A dimension attribute is an ordered pair (*dim*, *dd*), where *dim* and *dd* are the name and the domain of the attribute respectively and *dd* must be a DD.

Definition 9: A measurement attribute is an ordered pair (*mea*, *psd*), where *mea* and *psd* are the name and the probability stream domain of the attribute respectively and *psd* is a PSD.

Definition 10: ***Probabilistic Stream Schema (PS-schema)***

> A PS-schema is comprised of a non-empty list of dimension attributes and a list of measurement attributes.

A PS-schema is an ordered list of attribute name and domain pairs, analogous to a conventional relational schema. Attributes in a PS-schema are divided into two types, namely dimension attributes and measurement attributes. The values in dimension attributes are assumed to change only at some discrete time points and these changes are completely reflected on the tuples pertaining to the PS-schema. In contrast, values in measurement attributes may continuously change over time, in addition to their probabilistic nature. An associated prediction strategy gives hints on how measurement attribute values advance over time. The definition of PS-schema also requires that any valid PS-schema must have at least one dimension attribute, providing room for identifying streams.

Note that since a P-strategy is a model facilitating data sample predictions in a stream rather than a constraint on data in the stream, two PSDs are considered compatible if they have the same probabilistic data domain. Thus, two schemas are considered "compatible" as long as all of their corresponding attributes have compatible domains. When defining operations in this paper, we also use "same schema" to refer to compatible schema. The P-strategy changes are explicitly addressed.

Definition 11: ***Probabilistic Stream Tuple (PS-tuple)***

A PS-tuple of a PS-schema *Sch* contains a list of valid dimension attribute values, a list of valid measurement attribute values as determined by *Sch* or NULL, and a valid timestamp on universal clock.

A PS-tuple with NULL on all measurement attributes is also called NULL PS-tuple. The reason for inventing NULL PS-tuple will become clear when we introduce algebraic operations in the next section. In this paper NULL means *empty* or not *applicable* instead of *unknown*.

3.2 Probabilistic Stream Relation (PS-relation)

Definition 12: An object PS-tuple set *O* over a PS-chema *Sch* under a schedule *S* is a set of PS-tuples of *Sch* such that

1. all PS-tuples in *O* share the same values on all dimension attributes
2. the timestamp of each PS-tuple in *O* must be in *S*
3. each timestamp in *S* has exactly one PS-tuple in *O*.

An object PS-tuple set records the measures of an object (identified by the same dimension attribute values) at various time points as specified by a schedule.

Definition 13: ***PS-relation***

A PS-relation *R* over PS-schema *Sch* under a schedule *S* is the union of several object PS-tuple sets over *Sch* under *S*.

A PS-relation is a collection (possibly infinite number) of object PS-tuple sets compatible with the same schema. A PS-relation can have multiple object PS-tuple sets, each representing a series of observations of an observable identified by a distinct set of dimension attribute values. Each object PS-tuple set can be viewed as a sub-stream in the main stream represented by a PS-relation. All object PS-tuple sets in a PS-relation share the same PS-relation schedule and the same P-strategy for each attribute. All sub-streams in the same stream thus have strong logical ties, i.e. same schema, same schedule and same stream characteristics. The definition of the PS-relation also guarantees that a PS-relation is a clean view of observables because no redundant tuples are allowed and no unresolved observations of the same observable at the same time are allowed. PS-relations can be generated either directly from sensors (direct PS-relation) or derived through operations (derived PS-relation). We do not distinguish between these two types in our proposed framework as they pose no differences as far as the algebra (i.e., representation and processing) is concerned. An example PS-relation is shown in Appendix A.4.

3.3 Semantics of PS relation

A PS-relation represents all available information on the history of a number of observables. Each PS-tuple in the PS-relation represents a base fact observation of the observable at a particular time point With the predication

strategies available, observation at any time can be computed in a PS-relation. Thus the predication strategies in a PS-relation allows a possibly lossy representation of a continuous phenomenon with discrete finite number of tuples.

Equivalence of two PS-relations is no longer as simple as set membership testing in PSRA. Rather it is defined as information equivalence. For example, if R1 can be resampled to have same estimation quality (i.e. same uncertainty) at all time points as R2 have, R1 is information equivalent to R2. If estimation quality by R1 is same as R2 only on time points specified in a schedule S, R1 is information equivalent to R2 under S. Generally, less base fact observations are needed when a better prediction strategy is available to contain same amount of information. The concept of information equivalence can facilitate resource management in sensor networks. c. For example, given QoS requirement for a data stream, finding its PS-relation representation with minimum number of PS-tuples can reduce storage needed for the data stream and energy required to transmit data samples from one node to another

4. PS Operations

4.1 Helper operations

Cleaning

As mentioned in Section 2.2, cleaning functions, in the form of CL-strategies, are practically needed to remove data redundancies and to resolve observation inconsistencies, serving the purpose of constructing out of raw data a valid PS-relation.

Definition 14: *Cleaning*

Let T be a set of PS-tuples of the same PS-schema with a set C of CL-strategies, one for each measurement attribute. The cleaning of T using C , denoted $\kappa_C(T)$, is defined as follows:

$$\kappa_C(T) = \{(d, m, t) : \exists r \in T, r.time = t, d = r.\dim,$$
$$m = Clean(C, Measure(t, d, T))\}$$

where *r.time* and *r.dim* return the time and dimension attribute values of a PS-tuple *r* respectively, *Measure*(*t, d, T*) retrieves all measure attribute values of PS-tuples in *T* that have the same timestamp *t* and dimension attribute *d*, and *Clean*(*C, M*) applies the CL-strategies in *C* on the measure attribute values in *M*.

Cleaning of a PS tuple set creates a valid PS-relation with consistent observations and no data redundancy. The Cleaning operation first divides the PS tuple set T into subgroups by their dimension attributes values. For each subgroup (i.e. all tuples with same dimension attributes values), find all tuples with the same timestamp and apply the CL-strategy. Thus, after Cleaning operation, it is guaranteed that there is one and only one observation (PS tuple) for each distinct time point for each object tuple set. The set property of no duplicate tuples is automatically guaranteed by this operation. Example is omitted since this operation is straightforward.

Resampling

In sensor network applications, data points generated by sources may not match the data consumers' requirements, and thus resampling of the original streams is usually required. Semantically, resampling of a PS-relation generates a new view (specified by a new schedule) of observables from their current view.

Definition 15: *Resampling*

Let the PS-schema of a PS-relation R be $((dim_1, dd_1)\ldots (dim_m, dd_m), (mea_1, pdd_1)\ldots (mea_n, pdd_n))$ and the P-strategy associated with i'th measurement attribute be P_i. The resampling of a R under a schedule S, denoted as $\rho_S(R)$, is a PS-relation as defined below:

$$\rho_S(R) = \{(d_1,\ldots,d_m, m_1,\ldots,m_n, t) : t \in S \wedge \exists r \in R \wedge$$
$$r.\dim = (d_1,\ldots,d_m) \wedge r.m_1 = P_1(history_1, t) \wedge$$
$$r.m_2 = P_2(history_2, t) \wedge \ldots \wedge r.m_n = P_n(history_n, t)\}$$

where P_i denotes the prediction strategy for the i'th measurement attribute .

$history_i$ represents the history information passed in for prediction of attribute ζ_i. $history_i$ may come from within R or from other PS-relations.

Resampling of a PS-relation creates a new PS-relation with different tuple set, and thus a different view of the observables. The Resampling operation first divides the PS-tuple set R into subgroups by their dimension attributes values. Next, apply the P-strategy on values of each attribute in each subgroup (i.e. all tuples with the same dimension attributes values), to generate new values of the attribute for the subgroup at new time points as required in the schedule S. An illustrative resampling example is shown in Example 2 in Appendix A.4.

4.2 Stream Union

Union of two streams serves the purpose of merging two PS-relations with the same schema and producing a new PS-relation without duplicates. In applications, Union is used to merge observations about the same set of objects from different angles/paths to form a more complete view of the same set of objects. The resulting stream can have different observation time points from the input streams.

Definition 16: *Set Union*

The set union, denoted $\bigcup$, of two (compatible) PS-relations R_1 and R_2 is defined as

$$R_1 \bigcup R_2 = \{a \mid a \in R_1 \vee a \in R_2\}$$

The set union is very much like a conventional set union of the base tuple sets of R_1 and R_2. Note that the schemas of R_1 and R_2, though compatible, may have different P-strategies on corresponding attributes. The P-strategies in the resultant schema of the set union are left unspecified. Set union serves the purpose of defining the more general stream union.

Definition 17: *Stream Union*

The stream union of two PS-relations R_1 and R_2 using CL-strategy $\mathcal{C}$ under schedule S, denoted as $R_1 \bigcup_{(C,S)} R_2$, is

defined as follows:

1. The base tuple set in the result PS relation is given by

$$R_1 \cup_{(C,S)} R_2 = \kappa_c(\rho_S(R^{p1}) \cup \rho_S(R^{p2}))$$,where,

$$R^{p1} = \kappa_c(\rho_{(S1 \cup S2)}(R_1) \cup \rho_{(S1 \cup S2)}(R_2))$$ with

P-strategies same as those in R_1

$$R^{p2} = \kappa_c(\rho_{(S1 \cup S2)}(R_1) \cup \rho_{(S1 \cup S2)}(R_2))$$ with

P-strategies same as those in R_2

$S_1 \bigcup S_2$ denotes a schedule with all time points specified in S_1, the original schedule of R_1, and S_2, the original schedule of R_2

2. The P-strategy $\mathcal{P}$ for any measurement attribute a in the resultant PS-relation is specified by the following:

   $\mathcal{P}$(*history*,*t*)

   return $C(\mathcal{P}^1(history,t), \mathcal{P}^2(history,t))$

where

$\mathcal{P}^1$ denotes the P-strategy for attribute a as specified in R_1

$\mathcal{P}^2$ denotes the P-strategy for attribute a as specified in R_2

The stream union merges information in two PS-relations together. This operation is more complex than simple set union of all the tuples in the two PS-relations due to certain information, such as data correlation knowledge, is carried in the p-strategy. Thus, a complete stream union takes three steps. First all tuples from R_1 and R_2 are pooled together to form a larger base set, denoted as r. Then the p-strategy in R_1, denoted as p1, and the p-strategy in R_2, denoted as p2, are applied on the merged base set to estimate the observations at time points in the specified schedule. This step is done through the resampling operation in the above formula. After this step, two sets of observations at the same specified schedule points are generated. Finally, a cleaning operation is employed to combine the multiple estimations at each time point to generate a single consistent estimation on the schedule, which forms the data set of the result stream. The new p-strategy is also dynamically selected by the CL strategy.

The following subsections present the algebraic operations of PSRA, with examples shown in Appendix A.5.

4.3 **Stream Intersection**

The purpose of stream intersection of two PS-relations is to generate a stream with information that is verified by both streams. In practice, intersection can be used to find observations that have been verified from multiple sources and paths, i.e. the observations of the same object at the same time point from multiple sources/paths are consistent with each other. The consistency of multiple observations of a single attribute is determined by the similarity of the multiple observation p.d.f.'s.

Definition 18: ***Set Intersection***

The set intersection, denoted $\cap$, of two PS-relations R_1 and R_2, is defined as

$$R_1 \cap R_2 = \{a \mid (a \in R_1) \wedge (\exists (b \in R_2) \wedge (a.t = b.t) \wedge (a.\dim_1 = b.\dim_1) \wedge \ldots \wedge (a.\dim_m = b.\dim_m) \wedge Equal(a.mea_1, b.mea_1) \wedge \ldots \wedge Equal(a.mea_n, b.mea_n))\}$$

The set intersection matches up the tuples in R_1 and R_2 at the same time points. It is similar to conventional set intersection except for the equivalence of measurement attribute values (as defined by *Equal*()). *Equal*(m_1, m_2) is a system-defined predicate that determines whether the two p.d.f.'s.s m_1 and m_2 are equal. For discrete p.d.f.'s , an example difference function is cross-entropy, or called KL-distance [KL51]. Specifically, the cross entropy of m_1 to m_2 is defined as

$$D(m_1, m_2) = \sum_{x \in PDD} m_1(x) \cdot \lg \frac{m_1(x)}{m_2(x)},$$

where PDD is the data type of the probability data domain of m_1 and m_2. Thus, the equivalence predicate *Equal*(m_1, m_2) can be given by:

$$Equal(m_1, m_2) \equiv \max(D(m_1, m_2), D(m_2, m_1) \le \varepsilon,$$

where ε is a system-defined threshold.

For continous p.d.f's such as Gaussian p.d.f's, the equality measure is replaced by a *similarity* function as proposed in [FGB02].

Definition 19: ***Stream Intersection***

The stream intersection of two PS-relations R_1 and R_2 under schedule S, denoted $R_1 \cap_{(S)} R_2$, is defined as

1. The base tuple set in the resultant PS-relation is specified by the following:

   Let

   $O = \{d : (\exists a \in R_1) \wedge (\exists b \in R_2) \wedge (a.\dim = b.\dim) \wedge (d = a.\dim)\}$ and

   $R = \rho_S(R_1) \cap \rho_S(R_2)$. Here O denotes the set of objects appeared in both relations and R the set of observations on these objects agreed in both R_1 and R_2.

   $$R_1 \cap_{(S)} R_2 = \{(d, m, t) : (t \in S) \wedge (d \in O) \wedge ((\exists r \in R \wedge (r.t = t) \wedge (r.\dim = d) \wedge (r.mea = m)) \vee (\neg \exists r \in R \wedge (r.t = t) \wedge (r.\dim = d) \wedge (m = NULL)))\}$$

2. The P-strategy$\mathcal{P}$ for any measurement attribute a in the resultant PS-relation are specified by the following:

   $\mathcal{P}$(*history*,*t*)

   if $\mathcal{P}^1(history,t) = \mathcal{P}^2(history,t)$

   then return $\mathcal{P}^1(history,t)$

   else return NULL

where

$\mathcal{P}^1$ denotes the P-strategy for attribute a in R_1,

$\mathcal{P}^2$ denotes the P-strategy for attribute a in R_2, and

The stream intersection operation finds the information that is agreed to by multiple PS-relations. A stream intersection takes three steps. First, all tuples from R_1 are resampled to obtain estimations for the operator schedule S. Then all tuples from R_2 are resampled to obtain estimations for the operator schedule *S*. Now, at each schedule point we have two estimations, from R_1 and R_2 respectively. If these two estimations agree (based on p.d.f. equivalence when the attribute is probabilistic), the agreed estimation serves as the tuple at that time point in the result PS-relation. If not, a NULL tuple is created for that time point, meaning no estimation is available or the uncertainty is infinite. The p-strategy for the result PS-relation is constructed as a composition of the p-strategies in the input PS-relations following the logic described above.

4.4 **Stream Difference**

The difference of two PS-relations is used to eliminate certain observations as described in one relation from the other. In practice, this operation can be used to remove from a stream certain observations that are later deemed inappropriate. For example, readings in certain range from one sensor might need to be removed because it is found that the sensor was wrongly configured for that range. As another example, civilian users are allowed to use GPS with precision less than what is required by the military. Thus, certain tuples in the GPS data stream will need to be removed before being fed to civilian applications.

Definition 20: ***Set Difference***

The tuple set difference, denoted "–", of two PS-relations R_1 and R_2, is defined as

$$R_1 - R_2 = \{a \mid (a \in R_1) \wedge ((\neg \exists b \in R_2) \wedge (a.t = b.t) \wedge (a.\dim = b.\dim) \wedge Equal(a.mea, b.mea))\}$$

The set difference matches up the tuples in R_1 and R_2 at the same time points. If all attribute values of a tuple in R_1 agree with those of some timestamp-matching tuple in R_2, the tuple in R_1 is removed.

Definition 21: ***Stream Difference***

The stream difference of two PS-relations R_1 and R_2 under schedule *S*, denoted $R_1 -_{(S)} R_2$, is defined as

1. The base tuple set in the result PS-relation are specified by the following.
   Let $O = \{d : (\exists a \in R_1) \wedge (d = a.\dim)\}$ and $R = \rho_S(R_1) - \rho_S(R_2)$. Here *O* denotes the set of objects appeared in R_1 and *R* the set of observations in R_1 diagreed by R_2.

$$R_1 -_{(S)} R_2 = \{(d, m, t) : (t \in S) \wedge (d \in O) \wedge ((\exists r \in R \wedge r.t = t \wedge r.\dim = d \wedge r.mea = m) \vee (\neg \exists r \in R \wedge r.t = t \wedge r.\dim = d \wedge m = NULL))\}$$

2. The P-strategy $\mathcal{P}$ for any attribute *a* in the resultant PS-relation are specified by the following:
   $\mathcal{P}$(*history*,t)
   if $\mathcal{P}^1$(*history*,t) = $\mathcal{P}^2$(*history*,t)
   then return NULL
   else return $\mathcal{P}^1$(*history*,t)

where
$\mathcal{P}^1$ denotes the P-strategy for attribute *a* in R_1,
$\mathcal{P}^2$ denotes the P-strategy for attribute *a* in R_2, and

The stream difference operation removes information that is available in R_2 from R_1. A stream difference takes three steps. First, all tuples from R_1 are resampled to obtain estimations for the operator schedule S. Next, all tuples from R_2 are resampled to obtain estimations for the operator schedule S. Now, at each schedule point we have two estimations, from R_1 and R_2 respectively. If these two estimations agree (based on the p.d.f. equivalence when the attribute is probabilistic), the agreed estimation is removed by putting up a NULL tuple (meaning Not Applicable) for that time point in the result PS-relation. If not, the estimation tuple from R_1 is preserved as the tuple for that time point in the result PS-relation. The p-strategy in the result PS-relation is constructed as a composition of the P-strategies in the input PS-relations following the logic described above.

4.5 **Stream Select**

Definition 22: ***Stream Select***

The stream select of a PS-relations *R* over predicate *pr* under schedule *S*, denoted as $\sigma_{(pr,S)}R$, is defined as follows.

1. The base tuple set in the resultant PS-relation are specified by the following.
   Let $O = \{d : (\exists a \in R_1) \wedge (d = a.\dim)\}$. Here *O* denotes the set of objects appeared in *R*.

$$\sigma_{(pr,S)}R = \{(d, m, t) : (t \in S) \wedge (d \in O) \wedge (\exists r \in R \wedge ((pr(r) \wedge (m = r.mea)) \vee (\neg pr(r) \wedge (m = NULL))))\}$$

2. The P-strategy $\mathcal{P}$ for any measurement attribute *a* in the resultant PS-relation is specified as follows:
   $\mathcal{P}$(*history*,t)
   if *pr*($\mathcal{P}^{x1}$(*history*,t), …, $\mathcal{P}^{xk}$(*history*,t)) = TRUE
   then return $\mathcal{P}^a$(*history*,t)
   else return NA

where
x_i, $1 \le i \le k$, denote the attributes involved in evaluation of *pr*,
$\mathcal{P}^{xi}$ denotes the P-strategy for attribute x_i in *R*, and
$\mathcal{P}^a$ denote the P-strategy for attribute *a* in *R*.

Note that if some measurement attributes are included in the predicate *pr*, the computation of *pr* has to be probabilistic, following the basic operations described in Section 2.2. In addition, the result of the evaluation of *pr* is also probabilistic. For example, the evaluation of ($N(\mu=30,\sigma=2)<32$) results in a Boolean distribution. To deal with this problem, we assume a user-defined threshold associated with each selection operation which determines the minimum TRUE probability to satisfy the condition. For example, a 70% threshold will

make the evaluation of ($N(\mu=30,\sigma=2)<32$) TRUE, while a 90% threshold will result in FALSE for the evaluation of ($N(\mu=30,\sigma=2)<32$).

The stream select operation picks information that satisfies a specified condition. The stream select is carried out in two steps. First the original PS-relation *R* is resampled to obtain estimations for the operator schedule *S*. Then at each schedule point the predicate is evaluated against the estimation at that time point. If the predicate evaluates to TRUE, the estimation is passed to the output PS-relation. Otherwise, a NULL tuple for that time point is passed to the output PS-relation. The same logic is used to construct the output P-strategies out of the original P-strategies in the input PS-relation.

4.6 **Stream Projection**

The purpose of projection of a PS-relation is to produce a new PS-relation with only part of the attributes in a PS-schema from the original PS-relation, and to remove duplicate tuples. Note that the observation timestamp is not part of a PS-schema, and thus it can not be crossed out. Also, the elimination of certain dimension attributes may result in duplicated estimations at the same time point. Thus a cleaning operation is needed to make the final result a valid PS-relation.

Definition 23: ***Stream Projection***

The stream projection of a PS-relations *R* over a subset of the attributes specified in *R*'s schema: A_{proj} = (A_1, …A_k), using CL-strategy *C* and under schedule *S*, denoted $\pi_{(C,S,A_{proj})}(R)$, is defined as follows:

1. The base tuple set of $\pi_{(CL,S,A_{proj})}(R)$ is given by $\pi_{(CL,S,A_{proj})}(R) = \kappa_C(\pi_{A_{proj}}(\rho_S(R)))$,
   where $\pi_{A_{proj}}$ is the conventional project operation without duplication elimination.
2. The schema of $\pi_{(CL,S,A_{proj})}(R)$ is simply the collection of attributes A_{proj}, the P-strategy of each measurement attribute remains the same.

The stream projection is very similar to the conventional projection operation (i.e. only the columns of the attributes to be projected are preserved) except a resample operation is done first to match the output schedule requirement and a cleaning operation is done to remove duplicate observations. The p-strategy for each projected attribute is also preserved.

4.7 **Stream Cartesian Product**

Cartesian product is used to merge two PS-relations with possibly different schemas. The schema of the resulting PS-relation is a concatenation of the schemas of the two input PS-relations. In sensor network applications this operation can be used to merge different types of data streams. For example, the temperature readings stream and pressure readings stream of the same gas chamber can be merged together to form a complete view of the same object.

Definition 24: ***Stream Cartesian Product***

The stream Cartesian product of two PS-relations R_1 and R_2 under schedule *S*, denoted as $R_1 \times_S R_2$, is defined as following:

1. The base tuple set of the resultant PS-relation is given by
   $$R_1 \times_S R_2 = \{(p,t) \mid (p_1 \in \rho_S(R_1)) \wedge (p_2 \in \rho_S(R_2)) \wedge (t \in S) \wedge (p_1.t = t) \wedge (p_2.t = t) \wedge (p = (p_1, p_2))\}$$
2. The schema of $R_1 \times_S R_2$ contains the attributes coming from both R_1 and R_2.

The stream Cartesian product takes two steps. First, the input PS-relations–R_1 and R_2, are resampled over the schedule *S*. Next, tuples with the same timestamps are paired up and attributes from R_2 are concatenated to the attributes from R_1. Because of the resampling step over the same schedule, each tuple in R_2 is guaranteed to pair up with one and only one tuple in R_1 with the same timestamp. A tuple in the Cartesian product of two PS-relations represents a simultaneous observation of all attributes in the original input streams at a time point.

4.8 **Stream Join**

The stream join operation is defined as a composition of Cartesian product and select, similar to the join operation in conventional relational model. We take the theta join as the example. Equal join and natural join are its special cases.

Definition 25: ***Stream Join***

The stream join of PS-relations R_1 and R_2 with predicate *pr* under schedule *S*, denoted as $R_1 \bowtie_{(pr,S)} R_2$, is defined as $R_1 \bowtie_{(pr,S)} R_2 = \sigma_{(pr,S)}(R_1 \times_{(S)} R_2)$

Example for *stream join* is omitted as it is a straightforward composition of Cartesian product and select

4.9 **Stream Aggregation**

The PS-relational model can also be extended to support stream aggregations. A stream aggregation, like an aggregation in conventional relational model, needs a GROUP BY condition.

Definition 26: ***Stream Aggregation***

The stream aggregation of a PS-relations *R* with a set of grouping attributes *G* (specified as a set of attributes) and a set of aggregation functions *F* (one for each attribute to be aggregated) under the history constraint *hc* (a predicate specifying whether a tuple should participate in the aggregation) and a output schedule *S*, denoted as $\Im_{(F,hc,G,S)}R$, is defined as follows.

1. The PS-schema of $\Im_{(F,hc,G,S)}R$ has two types of attributes: attributes contained in the group condition *G* (all attributes in this category become dimension attributes in the new schema) and attributes resulted from

aggregation functions *F* (all attributes in this category keep their type in the new schema, i.e. dimension attributes are still dimensional and measurement attributes are still measurements).

2. The base tuple set of $\Im_{(F,G,S)}R$ has one tuple for each time point *t* in S.

$$\Im_{(F,hc,G,S)}R = \{(g, f, t) : (t \in S) \wedge (\exists r \in R') \wedge (r.g = g) \wedge f = F(t, R', g)\},$$

where $R' = \{r \mid r \in \rho_S(R) \wedge hc(r)\}$ denotes the PS-relation resampled from R under schedule *S* and satisfy history constraint *hc* and *F*(*t*, *R'*, *g*) denotes the application of the aggregate functions in *F* to the tuples in *R* with grouping attribute value being *g* and timestamp ≤ *t*.

Example 9 shown in Appendix 5 also demonstrates how sliding time window is implemented as a history constraint in PSRA. Since PSRA supports arbitrary predicates as history constraints, arbitrary window type can be supported. Note that this history constraint can also be embedded into P-strategies to screen out unwanted tuples since P-strategies can be dynamically changed by users. Thus other operations without explicit history constraint can also operate on a window of stream history instead of the default whole history. History constraint is explicitly specified in stream aggregation only for the purpose of convenience without adding extra modeling power.

5. Properties of PSRA

In this section we show the modeling power of PSRA through the following three theorems.

Theorem 1: ***Operators in PSRA are non- blocking***

Proof: All operations (except Cleaning) in PSRA are applied on PS-relations and have an associated operation schedule. Because every attribute in a PS-relation has a prediction strategy, for any time instant t, a PS tuple with that timestamp can be estimated from existing base tuples and corresponding prediction strategies, regardless of whether any PS tuple in the base tuple has timestamp t. Cleaning operation, as a helper function, is only applied on pre-resampled data as defined in Stream Union and Stream Projection. Thus regardless of the operation schedule requirement, a result PS tuple can always be computed for any time instant in the schedule without waiting for any information. New information (PS tuples) injecting into input PS-relations only improve the quality of the results (e.g. less standard variations for Gaussian p.d.f.'s) at future observation time instants. Thus all operators in PSRA are non-blocking.

Theorem 2: ***Existing Deterministic data steam models can be modeled in PSRA***

Proof: Arasu et al [ABW03] has proved that the CQL data steam model is less expressive than Aurora, similar to the TelegraphCQ model and more expressive than other existing deterministic data stream models as discussed in Related Work. Thus we only need to show that CQL and Aurora models can be modeled in PSRA. Using Lemma 2 and Lemma 3 proved in Appendix A.6, we conclude that this theorem holds.

6. Related Work

Lack of directly comparable probabilistic data stream models, research works in deterministic data stream models and data uncertainty models are generally related to this paper.

Data stream models

Studies on data stream models, in a generalized sense, can be traced back to materialized view maintenance [H87, SR88, GM95, JMS95], where updates to views are continuously computed based on updates to base tables on which the views were defined. Next, this problem was studied in the context of active databases, where rules are continuously evaluated driven by database activities [PD99]. Explicit data stream models were first introduced in [TG+92] and recently have been studied extensively [S96, CD+00, ABW, AC+03, CC+03, CJ+03]. The details of these models are discussed below:

The Chronicle Data Model (CDM) introduced in [JMS95] separated the concept of relation from that of stream (Chronicle, in their original term). The CDM supports the common relational operators and aggregations with incremental maintenance of materialized views. The data model in *Tapestry* system [TG+92] is essentially relational models supporting time-oriented queries. In *Tapestry* all data including stream data are stored in database and queried periodically and results are merged together to form the result *stream*. NiagaraCQ [CD+00] introduced intelligent grouping to optimize continuous queries over large amount of XML documents. Data model wise, it is similar to *Tapestry*. Gigascope [CJ+03] takes a stream-only approach. Selection, join, aggregation and merge (similar to union) operations are supported over streams. *Tribeca* [S96] is another pure stream model supporting window specification and a subset of relational operators and aggregations in form of *demux* and mux. TelegraphCQ [CC+03] proposed a window-based stream-to-stream data model, where, for every instant in time, relational operators are applied to windows of input data streams, resulting in a set of tuples associated with that instant in time. Aurora [AC+03] proposed a procedural framework for data stream processing. Its data model is also stream-to-stream, with the following seven operators: *Filter*, *Map*, *Union*, *Bsort*, *Aggregate*, *Join*, *Resample*.

CQL [ABW03] takes a similar approach to CDM in that it also separates the concept of relation from that of stream to take advantage of existing optimization techniques for traditional relational algebra. Streams can be converted to relations using windows. Relations can be converted to streams by generating new elements along time using *Isteam*, *Dstream* or *Rstream*

operators. Operations in CQL are carried out in three steps. All streams are first converted to relations. Then traditional relational operators are applied on these relations. Finally all result relations are converted back to result streams.

All existing data models discussed above treat a data stream as a bag of ordered tuples, either explicitly or implicitly, without consideration of inherent correlations within a stream and across streams. In addition, all of these models are deterministic in the following senses:

1. All data types are deterministic.
2. There is no formal model of data uncertainty incorporation and propagation.
3. There is no built-in probabilistic query support.

Data uncertainty models

Management of data uncertainty in database systems can be divided into probabilistic models [BG+92] and fuzzy models [AR84, KF88]. However fuzzy logic is more concerned with compensating for the lack of expressivity in a language instead of directly with data uncertainty [FGB02]. Thus probabilistic model is more suitable of modeling uncertainties in data from physical world [FGB02].

Research in probabilistic data model can be traced back to the study of data incompleteness in databases [W82, IL84]. Probabilistic data model supporting discrete p.d.f.'s have been extensively investigated [CP87, BG+92, DS96]. Faradjian et al [FGB02] recently proposed a Gaussian distribution abstract data type supporting continuous p.d.f. Faradjian generalized the equality of two Gaussian p.d.f's using similarity measurement and measure the difference of two Gaussian p.d.f's with *total variation distance* [FGB02]. However mutual independency are implicitly assumed in all operations in [FGB02] as no dependency model was discussed and employed in [FGB02]. Lakshmanan et al first introduced dependency models for computing probability of composite events [LL+97]. However their model only supports Boolean data type and is not first-order.

The probabilistic relational algebra for temporal databases proposed by Dekhtyar et al. [DRS01] has been particularly inspiring to our work. Their algebra supports modeling and reasoning of uncertainty about the start time, end time and duration of events while keeping all measurement values deterministic. In sensor data applications, we often need to model and reason about the uncertainty inherent in measurements at precise time instants as discussed in section 1.

7. Conclusions

Sensor data streams exhibit special characteristics such as inherent information uncertainty, inherent data sample correlations within and across streams, sensitive energy consumption and context-dependent data importance. We introduce the Probabilistic Stream Relational Algebra (PSRA), a data model for sensor data steams, by extending conventional relational model to address these characteristics. Specifically, a sensor data stream is modeled with a PS-relation consisting of a series of probabilistic data samples, capturing the inherent data uncertainties, and an associated prediction strategy capturing the inherent data sample correlations. Dependency models are introduced, in form of composition strategies, to support complex probabilistic operations. A predicate-based schedule specification is proposed, supporting specification of both push (e.g. event-driven) and pull (e.g. periodic query) type operations, to explicitly specify when to produce the next data tuple for an operation. With this model, memory required to store information of a data stream (i.e. memory required to store the base tuples for the stream) can be reduced when a high quality data correlation model is available. Data stream quality now can be intuitively defined by data uncertainties in desired schedule. Application QoS requirements can be expressed as data uncertainty requirements in its desired schedule. This requirement can then be transformed backward to guide the scheduling of underlying operations and even the sensors themselves. This schedule requirement propagation is a dynamic process, depending on the computing context. Thus PSRA provides a context-ware mechanism facilitating energy management in sensor networks through operator scheduling feedbacks. We also prove that operators in PSRA are non-blocking, thus making PSRA especially suitable for data stream processing. Finally we demonstrate the modeling power of PSRA by showing that conventional relational model and existing deterministic data stream processing models can be modeled in PSRA.

This paper represents our initial effort in building a probabilistic sensor data stream processing system. Future work includes study of equivalences in PSRA for execution optimizations and construction of a prototype system.

8. Acknowledgements

This work has been partially supported by the NSF under grant ISS-0308264, the ARDA agency under contract F30602-03-C-0243 and the Army High Performance Computing Research Center under contract DAAD19-01-2-0014. The content of this work does not necessarily reflect the position or policy of the government and no official endorsement should be inferred. Access to computing facilities was provided by the AHPCRC and the Minnesota Supercomputing Institute.

References

[ABW03] A. Arasu, S. Babu and J. Widom, "CQL: A Language for Continuous Queries over Streams and Relations," DBPL 2003: 1-19

[AC+03] D. Abadi, D. Carney et al., "Aurora: a new model and architecture for data stream management," VLDB Journal 2003

[AMS96] N. Alon, Y.Matias and M. Szegedy, “The space complexity of approximating the frequency moments,” Proc. Of the 1996 Annual ACM Symp. On Theory of Computing, Pg20-29, 1996

[AR84] M. Anvari and G. Rose, “Fuzzy relational databases,” Proc. Of the 1st International Conf on Fuzzy Information Processing, Pg B-6-3, CRC Press, 1984

[BBD+02] B. Babcok, S. Babu etc., “Models and Issues in Data Stream Systems,” PODS 2002, 1-16

[BG+92] D. Barbara, H. Garcia-Molina et al, “The Management of probabilistic data,” TKDE, 4(5):487-502, 1992

[BGS00] P. Bonnet, J. Gehrke and P. Seshadri, “Querying the Physical World,” IEEE Personal Communications, Vol 7 No 5, Oct. 2000, Pg10-15. Special Issue on Smart Spaces and Environments

[BL98] Y. Bar-Shalom and X. Li, “Estimation and Tracking: Principles, Techniques and Software,” YBS Publishing, 1998

[CC+02] D. Carney, U.Centintemel, et al. “Monitoring Streams – a new class of data management applications,” Proc. Of the 28th Intl. Conf. On Very Large Data Bases, pg 215-226, Aug. 2002.

[CC+03] S. Chandrasekharan, O. Cooper, et al. “TelegraphCQ: Continuous dataflow processing for an uncertain world,” Proc. Of the 1st Conf. On Innovative Data Systems Research, pg 269-280, Jan 2003.

[CD97] Surajit Chaudhuri, Umeshwar Dayal: An Overview of Data Warehousing and OLAP Technology. SIGMOD Record 26(1): 65-74 (1997)

[CD+00] J. Chen, D. DeWitt et al, “NiagaraCQ: A scalable continous query system for internet databases,” Proc. Of the 2000 ACM SIGMOD Intl. Conf. On Management of Data, pg379-390, May 2000

[CG+00] K. Chakrabarti, M. Garofalakis et al, “Approximate query processing using wavelets,” Proc. Of the 2000 Intl Conf. On Very Large Data Bases, Pg 111-122, 2000

[CJ+03] C. Cranor, T. Johnson et al., “Gigascope: A steram database for network applications,” Proc. Of 2003 ACM SIGMOD Intl. Conf. On Management of Data, pg647-651, June 2003

[CP87] R. Cavallo and M. Pittarelli, “The theory of probabilistic databases,” Proc. Of the Conf. On VerLarge Data Bases, 1987

[DRS01] A. Dekhtyar, R. Ross and V. Subrahmanian. “Probabilistic Temporal Databases, I: Algebra,” ACM Trans. On Database Systems, Vol.26, No.1, pg 41-95, March 2001

[DS96] D. Dey and S. Sarkar, “A probabilistic relational model and algebra,” TODS, 21(3):339-369, 1996

[EMS02] D. Estrin, M. Srivastava and A. Sayeed, Mobicom 2002 Tutorial T5 on Wireless Sensor Networks, Mobilcom, 2002

[FGB02] A. Faradjian, J. Gehrke and P. Bonnet, “GADT: A Probability Space ADT for Representing and Querying the Physical World” ICDE, 2002.

[GM95] A. Gupta and I.S. Mumick. Maintenance of materialized views: Problems, techniques, and applications. IEE Computer Society Bulletin of the Technical Comm. On Data Eng., 18(2):3-18, June 1995

[H87] E. Hanson, “A Performance Analysis of View Materialization Strategies,” SIGMOD Conference 1987: 440-453

[IL84] T. Imielinski and W. Lipski, “Incomplete information in relational databases,” Journal of the ACM, 31(4):761-791, Oct. 1984

[JK+98] H. Jagadish, N. Koudas et al, “Optimal histograms with quality guarantees,” Proc. Of the 1998 Intl Conf. On Very Large Data Bases, Pg 275-286, 1998

[JMS95] H. V. Jagadish, I. S. Mumick and A.Silberschatz, “View maintenance issues for the chronicle data model. Proc. Of the 14th ACM SIGACT-SIGMOD-SIGART symp. On Principles of Database Systems, pg 113-124, May 1995

[K60] R. Kalman, “A new approach to linear filtering and prediction problems,” Tran of the ASME-Journal of Basic Engineering, 82(Series D):35-45

[KF88] G. Klir and T. Folger, “Fuzzy sets, uncertainty and information, Prentice Hall, New Jersey, 1988

[KL51] .S. Kullback, R. A. Liebler, “On Information and Sufficiency,” Annals of Mathematical Statistics, 22, 76-86, 1951.

[LL+97] L. Lakshmanan, N. Leone et al, “ProbView: A flexible probabilistic database system,” TODS, 22(3):419-469, 1997

[M79] P. Maybeck, “Stochastic models, estimation, and control,” Vol 1, Academic Press, 1979

[M95] A. Mills, “Heat and mass transfer,” Burr Ridge, 1995

[PD99] N. Paton and O. Diaz, “Active Database systems,” ACM Computing Surveys, 31(1):63-103, March 1999

[R95] Y. Rozanov, “Probability theory, random processes, and mathematical statistics,” Kluwer Academic Publishers, 1995

[S96] M. Sullivan, “A stream database manager for network traffic analysis,” Proc. Of the 22nd Intl. Conf. On Very Large Data Bases, pg594, Sept. 1996

[SR88] J. Srivastava and D. Rotem, “ Analytical Modeling of Materialized View Maintenance,” PODS 1988: 126-134

[TG+92] D. Terry, D.Goldberg and et al., “continuous queries over append-only databases,” Proc. Pf the 1992 ACM SIGMOD Intl. Conf. On Management of Data, pg321-330, June 1992

[W82] Eugene Wong, “A statistical approach to incomplete information in database systems,” TODS, 7(3):470-488, 1982

[W94] C. Wright, “Applied Measurement Engineering: How to Design Effective Mechanical Measurement Systems,” Prentice Hall, 1994

Appendix

A.1. Definitions of CONST and IGNORANT Prediction Strategies

CONST Prediction Strategy:

Φ(*history*, *t*)
return $history@.t_0$

where

$history@.t_0$ denotes attribute value at time t_0 (t_0 <t)

IGNORANT Prediction Strategy:

Φ(*history, t*)
If there exists *history@.t*
then return *history@.t*
else return $history@.t^-$ with infinite uncertainty

where

t^- denotes the time instant most recent to *t* and available in *history*

A.2. Sample Composition Strategies

Table 1 shows composition strategies for Gaussian p.d.f's. Table 1 is derived from common practice in measurement engineering [W94]. Tables 2 shows composition strategies for Boolean random variables. Table 2 is derived from [LL+97] by adding conservative or aggressive constraints to maintain first order representations of probabilistic Boolean values.

Table 1: Composition Strategies for Gaussian p.d.f under algebraic operations

		+	-	*	/
Ignorance	Conservative	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \sigma_1+\sigma_2)$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \sigma_1+\sigma_2)$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \mid \mu_2*\sigma_1\mid+\mid\mu_1*\sigma_2\mid)$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \mid\sigma_1/\mu_2\mid+\mid\sigma_2/\mu_1\mid)$
	Aggressive	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \mid\sigma_1-\sigma_2\mid)$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \mid\sigma_1-\sigma_2\mid)$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \| \mu_2*\sigma_1\|-\|\mu_1*\sigma_2\|\|)$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \|\mid\sigma_1/\mu_2\mid-\mid\sigma_2/\mu_1\mid\|)$
Positive Correlation	Conservative	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \sigma_1+\sigma_2)$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \max(\sigma_1, \sigma_2))$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \mid \mu_2*\sigma_1+\mu_1*\sigma_2\mid)$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \mid\sigma_1/\mu_2+\sigma_2/\mu_1\mid)$
	Aggressive	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \min(\sigma_1, \sigma_2))$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \mid\sigma_1-\sigma_2\mid)$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \min(\mid\mu_2*\sigma_1\mid, \mid\mu_1*\sigma_2\mid))$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \min(\mid\sigma_1/\mu_2\mid,\mid\sigma_2/\mu_1\mid))$
Negative Correlation	Conservative	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \max(\sigma_1, \sigma_2))$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \sigma_1+\sigma_2)$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \max(\mid\mu_2*\sigma_1\mid, \mid\mu_1*\sigma_2\mid))$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \max(\mid\sigma_1/\mu_2\mid,\mid\sigma_2/\mu_1\mid))$
	Aggressive	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \mid\sigma_1-\sigma_2\mid)$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \min(\sigma_1, \sigma_2))$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \mid \mu_2*\sigma_1-\mu_1*\sigma_2\mid)$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \mid\sigma_1/\mu_2-\sigma_2/\mu_1\mid)$
Independence	Conservative Aggressive	$(\mu_1, \sigma_1) + (\mu_2, \sigma_2) = (\mu_1+\mu_2, \text{sqrt}(\sigma_1^2+\sigma_2^2))$	$(\mu_1, \sigma_1) - (\mu_2, \sigma_2) = (\mu_1-\mu_2, \text{sqrt}(\sigma_1^2+\sigma_2^2))$	$(\mu_1, \sigma_1) * (\mu_2, \sigma_2) = (\mu_1*\mu_2, \text{sqrt}((\mu_2*\sigma_1)^2+(\mu_1*\sigma_2)^2))$	$(\mu_1, \sigma_1) / (\mu_2, \sigma_2) = (\mu_1/\mu_2, \text{sqrt}((\sigma_1/\mu_2)^2+(\mu_1*\sigma_2/\mu_2^2)^2))$

Table 2: Composition Strategies for Boolean variable under logic operations[1]

		Conjunction ($\wedge$)	Disjunction ($\vee$)
Ignorance	Conservative(lower)	$\max(0, p_1+p_2-1)$	$\max(p_1, p_2)$
	Aggressive	$\min(p_1, p_2)$	$\min(1, p_1+p_2)$
Positive Correlation	Conservative	$\min(p_1, p_2)$	$\max(p_1, p_2)$
	Aggressive		
Negative Correlation	Conservative	$\max(0, p_1+p_2-1)$	$\min(1, p_1+p_2)$
	Aggressive		
Independence	Conservative	$p_1 * p_2$	$p_1+p_2-(p_1*p_2)$
	Aggressive		

A.3. *Predicate-based Schedule Specification (PSS)*

[1] NOT is an unary operator, thus not included in this table

A *predicate-based schedule specification*, PSS, for Schedule S is a mapping of $\{\{pt_1, \ldots, pt_n\}, \{st_1, \ldots, st_m\}\} \rightarrow t_o$, where

1. n and m are finite integers
2. pt_k $(1 \le k \le n)$ is a deterministic predicate. (i.e. it evaluates to either TRUE or FALSE)
3. st_k $(1 \le k \le n)$ is an internal state needed for predicate evaluation
4. t_o is the next time point in S whenever PSS is invoked

Schedules used in sensor network applications fall into three categories: absolute, relative and value-based. Absolute schedule is specified directly through a series of timestamps. For example, monitor temperature readings every 5 minutes from 00:00:00. Relative schedule is specified as time differences to other schedules. For example, whenever a temperature reading is received. Value-based schedule is determined by evaluation of the predicates over certain attribute values. For example, report temperature when it is greater than 50 degrees. A general schedule can be a combination of these three types. Thus computing next time point requires continuous evaluation of the predicates for these three types of schedules.

A.4. Illustrative Examples for PS-relations and resampling

Example 1: Consider a PS-schema S_T=(*SensorId*, *ObjMonitored*, *SensorLoc*, *Temperature*) for temperature sensors, where *SensorId*, *ObjMonitored*, *SensorLoc* are dimension attributes representing sensor id, object to be monitored, and location of the sensor, while *Temperature* is a measurement attribute. The following is an example PS-relation *T* over the PS-schema S_T under a schedule *S*=(1, 2, 3, 5).

SensorId	*ObjMonitored*	*SensorLoc*	*Temperature*	*Timestamp*
S1	O0001	(20, 50)	N(50, 0)	1
S1	O0001	(20, 50)	N(51, 1)	2
S1	O0001	(20, 50)	N(52, 1)	3
S1	O0001	(20, 50)	N(55, 0)	5
S2	O0001	(30, 60)	N(51, 1)	1
S2	O0001	(30, 60)	N(51, 0)	2
S2	O0001	(30, 60)	N(51, 1)	3
S2	O0001	(30, 60)	N(52, 0)	5

There are two object PS-tuple sets in *T*, identified by dimension attribute values ('S1', 'O0001', '(20, 50)'), and ('S2', 'O0001', '(30, 60)') respectively. The four grey tuples belong to object PS-tuple set ('S1', 'O0001', '(20, 50)'). The rest four tuples belong to object PS-tuple set ('S2', 'O0001', '(30, 60)'). They record the temperature readings measured by the two sensors in accordance with schedule *S*.

Example 2: Consider the input PS-relation as given in Example 1. Assume that S is {1,2,3,4,5}, the P-strategies for Temperature is:

$$\mu(t_k) = \mu(t_{k-1}), \sigma(t_k) = \sigma(t_{k-1}) + 1.0 \cdot e^{0.5 \cdot (t_k - t_{k-1})}$$

$\rho_S(R)$ yields:

SensorId	*ObjMonitored*	*SensorLoc*	*Temperature*	*TS*
S1	O0001	(20, 50)	N(50, 0)	1
S1	O0001	(20, 50)	N(51, 1)	2
S1	O0001	(20, 50)	N(52, 1)	3
S1	O0001	(20, 50)	N(52, 1+$\sqrt{e}$)	4
S1	O0001	(20, 50)	N(55, 0)	5
S2	O0001	(30, 60)	N(51, 1)	1
S2	O0001	(30, 60)	N(51, 0)	2
S2	O0001	(30, 60)	N(51, 1)	3
S2	O0001	(30, 60)	N(51, 1+$\sqrt{e}$)	4
S2	O0001	(30, 60)	N(52, 0)	5

Note that in addition to prediction, P-strategies can be used to model sliding windows. For example, one can specify a P-strategy, which only returns data with timestamp in the most recent one hour (return NULL for all other timestamps), to simulate a sliding time window as commonly used in other data stream models.

A.5. Illustrative Examples for PSRA Operators

Example 3 Consider two PS-relations T_1 and T_2 over the same PS-schema S_T=(*ObjMonitored*, *Temperature*), where *ObjMonitored* is a dimension attribute and *Temperature* is a measurement attribute. In addition, the P-strategies of *Temperature* in T_1 and T_2 are given by

P-strategy for T_1: $\mu(t_k) = \mu(t_{k-1}), \sigma(t_k) = \sigma(t_{k-1}) + 1.0 \cdot e^{0.5 \cdot (t_k - t_{k-1})}$

P-strategy for T_2: $\mu(t_k) = \mu(t_{k-1}), \sigma(t_k) = \sigma(t_{k-1}) + 0.5 \cdot e^{0.5 \cdot (t_k - t_{k-1})}$

where $\mu(t_{k-1})$ and $\sigma(t_{k-1})$ are the mean and standard deviation of the Gaussian p.d.f. at time t_{k-1}.

T_1 (with schedule (1, 3, 5))

ObjMonitored	*Temperature*	*Timestamp*
O0001	N(50, 0)	1
O0001	N(52, 1)	3
O0001	N(55, 0)	5
O0002	N(51, 1)	1
O0002	N(51, 1)	3
O0002	N(52, 0)	5

T_2 (with schedule (1, 2, 5))

ObjMonitored	*Temperature*	*Timestamp*
O0001	N(50, 0)	1
O0001	N(52, 1)	2
O0001	N(54, 0)	5

Let the CL-strategy *C*=***Average*** using conservative ignorance[2] and the schedule *S*=(1, 2, 3, 4, 5). The result PS-relation $T_1 \cup_{(C,S)} T_2$ is given in the following table:

ObjMonitored	*Temperature*	*Timestamp*
O0001	N(50, 0)	1
O0001	N(52, 1)	2
O0001	N(52, 1)	3
O0001	N(52, $1+0.75\sqrt{e}$)	4
O0001	N(54.5, 0)	5
O0002	N(51, 1)	1
O0002	N(51, $1+0.75\sqrt{e}$)	2
O0002	N(51, 1)	3
O0002	N(51, $1+0.75\sqrt{e}$)	4
O0002	N(52, 0)	5

And the new P-strategy on *Temperature* becomes

$\mu(t_k) = \mu(t_{k-1}), \sigma(t_k) = \sigma(t_{k-1}) + 0.75 \cdot e^{0.5 \cdot (t_k - t_{k-1})}$

Example 4. Consider the same PS-relations T_1 and T_2 as described in Example 3. Assume the p.d.f. equivalence threshold ε is set to be 0. The PS-relation output by $T_1 \cap_{C,S} T_2$, where *C*=**Average** and the schedule *S*=(1, 2, 3, 4, 5), is the following:

ObjMonitored	*Temperature*	*Timestamp*
O0001	N(50, 0)	1
O0001	NULL	2
O0001	NULL	3
O0001	NULL	4
O0001	NULL	5

Example 5. Consider the same PS-relations T_1 and T_2 as described in Example 3. Assume the p.d.f. equivalence threshold ε is set to be 0. The PS-relation output by $T_1 -_{C,S} T_2$, where *C*=**Average** and the schedule *S*=(1, 2, 3, 4, 5), is the following:

ObjMonitored	*Temperature*	*Timestamp*

[2] That is, $C((\mu_1, \sigma_1), (\mu_2, \sigma_2)) = \frac{\mu_1 + \mu 2}{2}, \frac{\sigma_1 + \sigma 2}{2}$.

O0001	NULL	1
O0001	NULL	2
O0001	N(52, 1)	3
O0001	N(52, $1+\sqrt{e}$)	4
O0001	N(55, 0)	5
O0002	N(51, 1)	1
O0002	N(51, $1+\sqrt{e}$)	2
O0002	N(51, 1)	3
O0002	N(51, $1+\sqrt{e}$)	4
O0002	N(52, 0)	5

Example 6. Consider the same PS-relations T_1 described in Example 3. The following shows the PS-relation output by $\sigma_{(Temperature<54,\ S)}T_1$, where S=(1, 2, 3, 4, 5) and the threshold assigning to minimum TRUE probability of the predicate *Temperature*<54 is 80%.

ObjMonitored	*Temperature*	*Timestamp*
O0001	N(50, 0)	1
O0001	N(50, $\sqrt{e}$)	2
O0001	N(52, 1)	3
O0001	NULL	4
O0001	NULL	5
O0002	N(51, 1)	1
O0002	N(51, $1+\sqrt{e}$)	2
O0002	N(51, 1)	3
O0002	N(51, $1+\sqrt{e}$)	4
O0002	N(52, 0)	5

Example 7. Consider the PS-relation T shown in Example 1. Assume that the P-strategy of *Temperature* is given by $\mu(t_k)=\mu(t_{k-1}), \sigma(t_k)=\sigma(t_{k-1})+1.0\cdot e^{0.5\cdot(t_k-t_{k-1})}$. Note that in every time point in T, there are two PS-tuples with same value in *ObjMonitored*. Thus, the project of *ObjMonitored* and *Temperature* from T requires cleaning. Let the CL-strategy C=***Average*** using conservative ignorance and the schedule S=(1, 2, 3, 4, 5). The PS-relation output by $\pi_{C,\ S,\ (ObjMonitored,\ Temperature)}T$ is the following:

ObjMonitored	*Temperature*	*Timestamp*
O0001	N(50.5, 0.5)	1
O0001	N(51, 0.5)	2
O0001	N(51.5,1)	3
O0001	N(51.5, $1+\sqrt{e}$)	4
O0001	N(53.5, 0)	5

Example 8. Consider the PS-relation T shown in Example 1 and another PS-relation P shown below:

SensorId	*ObjMonitored*	*Pressure*	*Timestamp*
P1	O0001	N(100, 0)	1
P1	O0001	N(104, 1)	2
P1	O0001	N(107, 1)	4
P1	O0001	N(110, 0)	5

Assume that the P-strategies for Temperature in T and Pressure in P are the following:
P-strategy for T: $\mu(t_k)=\mu(t_{k-1}), \sigma(t_k)=\sigma(t_{k-1})+1.0\cdot e^{0.5\cdot(t_k-t_{k-1})}$
P-strategy for P: $\mu(t_k)=\mu(t_{k-1}), \sigma(t_k)=\sigma(t_{k-1})+0.5\cdot e^{0.5\cdot(t_k-t_{k-1})}$

The PS-relation of $T\times_S P$, where S=(1, 2, 3, 4, 5) is the following:

T.SensorId	*T.ObjMonitored*	*T.SensorLoc*	*P.SensorId*	*P.ObjMonitored*	*Temperature*	*Pressure*	*Timestamp*
S1	O0001	(20, 50)	P1	O0001	N(50, 0)	N(100, 0)	1
S1	O0001	(20, 50)	P1	O0001	N(51, 1)	N(104, 1)	2
S1	O0001	(20, 50)	P1	O0001	N(52, 1)	N(104, $1+\frac{\sqrt{e}}{2}$)	3
S1	O0001	(20, 50)	P1	O0001	N(52, $1+\sqrt{e}$)	N(107, 1)	4
S1	O0001	(20, 50)	P1	O0001	N(55, 0)	N(110, 0)	5
S2	O0001	(30, 60)	P1	O0001	N(51, 1)	N(100, 0)	1
S2	O0001	(30, 60)	P1	O0001	N(51, 0)	N(104, 1)	2
S2	O0001	(30, 60)	P1	O0001	N(51, 1)	N(104, $1+\frac{\sqrt{e}}{2}$)	3
S2	O0001	(30, 60)	P1	O0001	N(51, $1+\sqrt{e}$)	N(107, 1)	4
S2	O0001	(30, 60)	P1	O0001	N(52, 0)	N(110, 0)	5

Example 9. Assume that schedule *S* is {1, 2}, Group condition is SensorId, F for temperature reading is AVG, R is given as follows after resampling, and independent composition strategy is adopted for example

SensorId	*Temperature reading*	*Timestamp*
Sensor01	N(50, 4)	1
Sensor01	N(40, 2)	2
Sensor02	N(60, 10)	1
Sensor02	N(50, 8)	2

a). When *hc* allows the entire history to participlate aggragation (i.e. hc = TRUE), $\Im_{(F,G,S)}R$ yields

SensorId	*Temperature reading*	*Timestamp*
Sensor01	N(50, 4)	1
Sensor01	N(45, 2.24)	2
Sensor02	N(60, 10)	1
Sensor02	N(55, 6.4)	2

b). When *hc* imposes a window constraint of length 1 by checking (current time – t < 1),

$\Im_{(F,G,S)}R$ yields

SensorId	*Temperature reading*	*Timestamp*
Sensor01	N(50, 4)	1
Sensor01	N(40, 2)	2
Sensor02	N(60, 10)	1
Sensor02	N(50, 8)	2

A.6. Lemmas for PSRA properties

Lemma 1: ***Traditional relational algebra (TRA) is a special case of PSRA***

Proof: The deterministic and non-stream data types generally used in TRA can be modeled by DD domain in PSRA framework. All attributes in TRA thus can be modeled as dimension attributes in PSRA. The CONST prediction strategy makes sure that only one tuple is needed for each dimension value for the whole history. Thus a PS-relation represents a traditional relation with the only extra information of the timestamp attribute. Now consider the stream operators when all attributes are dimensional. In this case the Cleaning and resampling operators always preserve the original relation since no data redundancy in original relation and all attributes have CONST P-strategy. Consequently:

stream union, *stream intersection* and *stream difference* are reduced to corresponding set operations

stream select becomes predicate evaluation on each dimension value, thus on each tuple in PS-relation

stream project becomes simple dimension attributes projection and resulting PS-relation also has only dimension attributes, thus a valid model of traditional relation

stream Cartesian product becomes pairing up every possible combination of dimension attribute values from input relations.

stream join has the same compositional definition as in traditional relational model. If *stream select* and *stream Cartesian product* are same as in TRA, *stream join* must be the same as the *join* operator in TRA.

stream aggregation becomes application of aggregation functions on dimensional values based on grouping conditions, regardless of the schedule. Result PS-relations are also all dimensional.

Therefore TRA is a special case of PSRA where all attributes are modeled as dimension attributes.

Lemma 2: ***CQL can be modeled in PSRA***

Proof: All deterministic models process information purely upon the timely availability of data. Deterministic models implicitly employ a black-white data uncertainty strategy: if a measurement has been received when it is needed, the is zero, otherwise, data uncertainty is infinite. Thus deterministic data streams can be generally modeled by PS-relations with IGNORANT prediction-strategy for all measurement attributes.

CQL [ABW03] can be considered a TRA extended with three window type operations (time-based, tuple-based and partitioned). Streams are first converted to relations using these window operations. TRA operators are then applied on the relation pool. Finally results of continuous queries are converted back to streams from relations. The window-based operations are modeled by the IGNORANT P-strategy with constraints on the data available in the input parameter *history*. For example, a simple 30 minutes sliding window in CQL can be modeled as a IGNORANT P-strategy where only the most recent 30 minutes data (i.e. data whose timestamps > (current time - 30 min) are passed to the IGNORANT P-strategy as the parameter *history*. Thus in PSRA, window specification can be modeled as input data constraints to P-strategies or having this check up implemented inside the IGNORANT P-strategy. Since PSRA support arbitrary P-Strategy, CQL window specification can be modeled in PSRA. Lemma 1 has shown that all TRA operators can be modeled in PSRA when dealing with pure non-stream relations. We conclude that CQL can be modeled in PSRA.

Lemma 3: ***Aurora Data steam model [AC+03] can be modeled in PSRA***

Aurora [AC+03] employs procedural stream-to-stream data model with the seven operators: *Filter*, *Map*, *Union*, *Bsort*, *Aggregate*, *Join*, *Resample*. We only discuss the its data modeling power without touch on resource management related actions. In the following discussion, P-strategies are always assumed to be IGNORANT unless explicitly stated.

Filter takes one input stream and generate (m+1) output streams, where m is the number of predicates to be evaluated. *Filter* can be modeled as m *stream select* operations with output schedule same as input schedule.

Map can be modeled by stream aggregation where attributes in the grouping condition include all dimension attributes and the timestamp.

Union can be readily modeled by the *stream union*.

BSort can be modeled by the *resampling* in PSRA with a P-strategy always returning minimal value in its computation buffer.

Aggregate can be modeled by the *stream aggregation* with the window identifying attributes set as dimension attributes.

Join in Aurora is essentially a window-based join and can be modeled as stream join with IGNORANT window-constrained P-strategy as discussed in the CQL part.

Resample can be readily modeled by the *resampling* operator in PSRA with P-strategy corresponding to F and schedule corresponding to the heartbeats. Though syntactically and functionally the Aurora *resample* is very similar to the PSRA *resampling*, we draw the distinction that no uncertainties introduced by estimations are considered in Aurora *resample*.

We thus conclude that data stream model in Aurora can be modeled in PSRA.