Efficiently Maintaining Distributed Model-Based Views on Real-Time Data Streams

Alexandru Arion, Hoyoung Jeung, Karl Aberer, Sebastian Michel

Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland
{alexandru.arion, hoyoung.jeung, karl.aberer, sebastian.michel}@epfl.ch

***Abstract*— Minimizing communication cost is a fundamental problem in large-scale federated sensor networks. Existing solutions applicable for the problem are often ad-hoc for specific query types, or they are inefficient when query results contain large volumes of data to be transferred over the networks. Maintaining model-based views of data streams has been recently highlighted because it permits the data communication over networks to be efficient by transmitting parameter values for the models, instead of sending original data streams. This paper proposes a novel framework that employs the advantages of using model-based views for communication-efficient stream data processing over federated sensor networks, yet it significantly improves state-of-the-art approaches. The framework is generic and any time-parameterized models can be plugged, as well as accuracy guarantees for query results are ensured throughout the large-scale networks. In addition, we boost the performance of the framework by the *coded model update* that enables efficient model update from one node to another. It predetermines parameter values for the model, updates only identifiers of the parameter values, and compresses the identifiers by utilizing bitmaps. Moreover, we propose a novel correlation model, named *coded inter-variable model*, that integrates the efficiency of the coded model update into more precise predictions of correlated models. Empirical studies with real data demonstrate that our proposal achieves substantial amounts of communication reduction, outperforming a state-of-the art method.**

I. Introduction

Although the Sensor Internet is still in its infancy, large numbers of sensor networks are already being interconnected and share huge amounts of streaming data from sensors. In the SwissEx project [1], [2], for example, various research institutes share real-time environmental observation data across many different local sensor networks; they become producers and consumers of streaming data simultaneously. In these *federated sensor networks*, data streams from a producer node are continuously delivered to multiple consumer nodes in different local networks. Minimizing communication cost over the networks has become a primary challenge for researchers.

There exists a rich body of research work on this problem, including communication-efficient data dissemination [3], [4], [5] and effective operator placement in distributed stream processing systems [6], [7], [8], [9]. Although these proposals reduce large amounts of data transfers over distributed sensor networks, they are often applicable only for particular query types, or inefficient when a query result contains a large volume of data.

This paper proposes a novel framework that is fundamentally different from the existing approaches. It employs (mathematical) models for representing data streams at producer nodes, and duplicates the models onto consumer nodes who need the data streams. Queries at the consumer nodes are processed directly over the data streams generated by the models, so-called *model-based views* [10], without fetching the actual data from the producer nodes. The framework then subsequently updates the models, so that real-time sensor readings are precisely captured by the models.

Model-based views have been introduced to achieve synergy among data processing using models and powerful data management functionalities provided by databases, the both tasks are often needed for applications but performed separately. In this paper, we go beyond this approach and present a framework adopting the model-based views to lead to various advantages for processing distributed data streams. The key features of the framework are briefly highlighted as follows:

- First, it permits to reduce the communication cost over networks significantly since it does not require any data transfer of actual streams. Only the parameters of models are updated through the networks. Typically, the model update occurs infrequently and yields substantially smaller amounts of data to be transmitted.
- Second, any type of queries with respect to the data streams can be processed at consumer nodes without sending queries and receiving the results across the networks, since the consumer nodes have all data required for the query processing, i.e., model-based views. This also prevents from potential network delay and data loss.
- Third, our framework is generic and any type of model can be employed, as long as the model is time-parameterized and able to predict the value at a given (current or future) time. This is important because a number of models are being used for different purposes in a wide range of applications. The framework allows such applications using their specific models, while efficient data communication over distributed networks is facilitated by simply plugging the models into the framework.
- Finally, the framework provides a systematic solution that can guarantee user-specified accuracy requirements for model-based views. The guarantees are independent of the model types used within the framework.

In fact, some prior work [11], [12], [13] has already utilized models for reducing data communication for stream processing over networks. Nevertheless, their methods are designed for specific models, while we aim to provide a generic platform that accepts arbitrary models used in federated sensor networks.

Furthermore, we propose two novel mechanisms. First, we introduce a *coded model update* that enables model update from one node to another very efficiently. The coded model update predetermines parameter values for a model, which are shared by both producer and consumer nodes. It then sends merely bitmap-encoded identifers of the parameters when the model update is required, instead of sending the actual parameter values for the model. We present concrete solutions for presetting the parameter values, guaranteeing user-provided error bounds, and encoding bitmaps.

Second, we propose a new model, called *coded inter-variable model*, using correlation information of multiple streams, which can compensate errors and noises of raw data by exploiting the dependencies from one stream to another. This yields more precise value prediction and reduces data redundancy, resulting in infrequent model update. Since our coded model update is designed to support any given model, we embody this correlation model based on the coded model update. Therefore, it brings synergy effects from combining the effectiveness of the correlation model and the efficiency of the coded model update.

The contributions of the paper are summarized as follows:

- We propose a novel, generic, and accuracy-guaranteed framework that utilizes model-based views for efficient data communication over federated sensor networks.
- We present how to fit various regression models into the framework as case studies, by showing a generic algorithm for adaptively computing the models.
- We introduce the coded model update as a generic compression scheme of model parameters for efficient model update over networks.
- We introduce the coded inter-variable model based on the coded model update method, which takes into account correlation information of multiple streams, resulting in more precise value predictions.
- We demonstrate comprehensive experimental analyses for our approach using two different real datasets.

The remainder of the paper is organized as follows. Section II presents our network model and the architecture of the framework. Section III provides a case study of plugging various regression models into our framework. Section IV introduces the coded model update and Section V describes the details of the coded inter-variable model. Section VI presents extensive experimental results for large, real sensor datasets. Section VII discusses the relevant works to our study, followed by conclusions in Section VIII.

Symbol	Meaning
N	node in federated sensor networks
v_t	raw sensor reading value at time t
s, s'	raw data stream, model-based view of s
ϵ_s	user-specified error bound for stream s
$\mathcal{M}, \mathcal{M}_s, \mathcal{M}_{.i}$	model, model for s, i-th model instance of $\mathcal{M}$
$v'_t, \mathcal{M}(t)$	model-driven value at time t
$\mathcal{M}^*$	most similar predetermined model to $\mathcal{M}$
$\mathcal{M}^\circ$	coded inter-variable model
$v^*_t, \mathcal{M}^*(t)$	model-driven value at time t, that generated by $\mathcal{M}^*$
$P = \{p_i\}$	set of parameters p_i required for building a model
$\bar{v}$, V_{p_i}	parameter value, set of parameter values for p_i
B_{p_i}, B_v	upper and lower bounds of $\bar{v}$ for p_i, those of $v \in s$
h	the number of subspaces in B_{p_i} or B_v
$o_i, \lvert o_i \rvert$	i-th slot of a bitmap, the number of bits in o_i

TABLE I
SUMMARY OF NOTATIONS

II. THE FRAMEWORK

In this section, we introduce our framework that utilizes model-based views for efficient data communication over federated sensor networks. Table I offers the notations that will be used throughout the paper.

A. Overview

Let $s = \langle v_1, v_2, \cdots, v_n \rangle$ be a raw data stream from one sensor, represented by a sequence of timestamped sensor readings, where $v_t \in s$ indicates the value at time t. We formally define the network model that this study considers:

Definition 1: A **federated sensor network** consists of a set of interconnected nodes $\{N_1, N_2, \cdots, N_a\}$, such that each node maintains a set of data streams $N_j = \{s_1, s_2, \cdots, s_b\}$, $j \in [1, a]$ in a local sensor network.

Figure 1 illustrates an example of a federated sensor network, connecting three local sensor networks with the corresponding nodes N_1, N_2, and N_3. In the example, node N_1 sustains two data streams s_1 and s_2 from two different sensors respectively, and the node is connected to nodes N_2 and N_3 through the Internet. Similarly, N_2 and N_3 maintain their own local streams in different local networks.

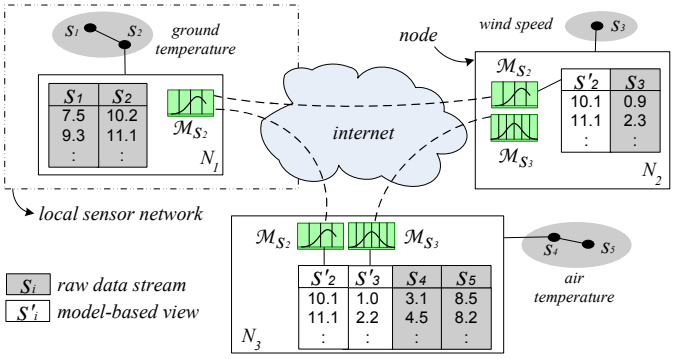

Fig. 1. An Example of The Framework over A Federated Sensor Network

In a federated sensor network, it often occurs that a node needs to bring the data streams maintained by another node (or multiple other nodes) across the Internet, for analyzing the data or processing queries. We call the former node a *consumer*

node and the latter node a *producer node*. It is also possible that the producer node needs the data streams maintained by the consumer node, and thus the both nodes become producers and consumers simultaneously.

Because transmitting raw data streams over a federated network requires high data communication cost, some previous studies [3], [4], [5] propose efficient approaches, however, they are still insufficient when the number of nodes in the network becomes large and the data streams are high-rate. Instead of sending the raw data streams, queries at the consumer nodes can be sent to the producer nodes [14], [8], [9], yet this approach is generally applicable only for particular query types or inefficient if query results contain large amounts of data.

To overcome the above shortcomings, our framework takes an essentially different approach—utilizing *model-based views* for efficient distributed data stream processing. It consists of the following two phases:

- *Initialization phase*: when a consumer node requests a data stream from a producer node, the producer node constructs a (time-parameterized) model that can predict the values at current or future times. The producer node then sends the model's parameters to the consumer and the consumer node builds the same model using the parameters received. This operation is performed only once when the connection between the two nodes is initially established.
- *Running phase*: the consumer node generates a model-based view as the representation of the raw stream by the model constructed. Any query at the consumer node is directly processed over the model-based view, without any data communication with the producer node. The framework subsequently updates the model's parameters from the producer node to the consumer node only when necessary.

Fig. 1 shows an example of these processes. N_2 (i.e., consumer node) asks stream s_2 from N_1 (i.e. producer node), and then model $\mathcal{M}_{s_2}$ is constructed at N_1. Next, N_1 sends the parameter values for $\mathcal{M}_{s_2}$ to N_2, and $\mathcal{M}_{s_2}$ built by the parameters received at N_2 generates model-based view s_2' for representing s_2. With the same manners, model-based views s_2' and s_3' at N_3 are also generated by $\mathcal{M}_{s_2}$ and $\mathcal{M}_{s_3}$, respectively.

B. Accuracy Guarantee and Model Update

When the framework constructs a model at a producer node, it takes a user-specified error bound, such that the difference between a raw sensor reading and its corresponding value in a model-based view, termed *model-driven value*, must not exceed the bound. Our framework provides this accuracy guarantee throughout the federated sensor network, formally stated as:

Property 1: Let $s = \langle v_1, v_2, \cdots, v_n \rangle$ be a raw data stream and $s' = \langle v_1', v_2', \cdots, v_n' \rangle$ be a model-based view created by model $\mathcal{M}_s$. Given an accuracy bound ϵ_s for s, the framework guarantees that

$$|v_t - v_t'| \leq \epsilon_s \qquad v_t \in s, v_t' \in s'$$

To maintain Property 1, the framework performs *model update* from a producer node to consumer nodes using the following steps: (i) the producer node generates a model-driven value when a new raw reading is streamed, and checks whether the difference between the raw value and the model-driven value stays within the error bound. (ii) If the difference does not exceed the error bound, no communication is required between the two nodes, and the consumer node generates values for their model-based views. (iii) Otherwise, the producer node reconstructs its model, so that the model-driven value generated from the reconstructed model does not exceed the error bound from the current raw reading. Next, the producer node updates the models at consumer nodes by sending new parameter values of the reconstructed model.

In some cases, multiple consumer nodes may set different error bounds to a single raw stream at a producer node. For such a case, we provide two options:

- *Multi-model update* maintains n models at the producer node for given n error bounds requested from consumer nodes, such that each model is updated according to one of the bounds. This may incur high computing overhead for the producer node, since the producer node is responsible for the model's reconstruction and update. Nevertheless, this option achieves minimum communication cost over the networks.
- *Single-model update* is performed according to only the smallest error bound among the n error bounds for all model updates, because the tightest error bound covers all the accuracies demanded from consumer nodes. Since the producer node maintains only one model, the cost for model maintenance is very low. On the other hand, the model update occurs more often because model reconstructions are performed frequently to keep the smallest error bound, which is unnecessary for larger error bounds.

C. The Hitchhiker's Guide to Model Selection

Our framework is generic and allows any type of models to be used, as long as the model is time-parameterized thus can predict a value at a given (current or future) time. Nevertheless, there are some key factors that influence the performance of the framework, which applications should consider for the model selection. We here provide some guidelines.

Be aware of what others have done. The literature provides a rich set of mathematical models that are widely used for processing streaming data, such as classic regression models [15], [16], Chebyshev approximation [17], piecewise constant approximation [18], correlation models [19], [20], Kalman fiters [11], hidden Markov models [21], [22], and dynamic probabilistic models [12]. This paper also provides a novel model in Section V. All the above models can be fit to the framework.

Precise prediction is most important. The model should predict a value v' as similar as possible to the corresponding raw reading v, since a large $|v - v'|$ is likely to exceed a given error bound, resulting in more frequent model update.

Make sure that prediction process is efficient. The value prediction by a model is performed at both producer and consumer nodes every time when a new sensor reading is streamed. Thus, it becomes costly if the stream has a high rate, and a large number of such streams are maintained by the nodes.

Do not neglect simple models. It is often observed that simple models such as piecewise constant and linear models do not perform worse than sophisticated models in terms of prediction accuracy. Moreover, simple models generally have low computing costs for their construction as well as small numbers of parameters to be sent for model update.

III. Case Study: Plugging Regression Models

In this section, we first briefly review various regression models widely used for approximating time-series data. We then describe how to fit the models to our framework as a case study.

A. Time–Series Regression

Given a stream $s = \langle v_1, v_2, \cdots, v_n \rangle$, regression models describe how values $v_t \in s$ vary depended on the passage of time $t \in [1, n]$. Our goal here is to predict a value v'_{n+1} using the regression models when a new sensor reading v_{n+1} is streamed, so that the framework can check whether $|v_{n+1} - v'_{n+1}|$ satisfies a given error bound.

We briefly introduce two popular types of regression models, i.e., *polynomial* and *Chebyshev* regressions.

Polynomial Regression.
Given a degree d, polynomial regressions find the best-fitting curve (or line when $d = 0$ or $d = 1$) that minimizes the total difference between the curve and each value v_t, formally defined as:

$$v_t = c + \alpha_1 \cdot t + \cdots + \alpha_d \cdot t^d, \qquad t = 1, ..., n \qquad (1)$$

, where c is a constant and α_j are regression coefficients.

The polynomial regressions with high degrees approximate the given stream with more sophisticated curves, rendering theoretically more accurate predictions. Practically, however, low-degree polynomials, such as constant ($d = 0$) and linear ($d = 1$), can also perform well because they compensate errors in real sensor data by simple regression lines. In addition, low-degree polynomial regressions are easy to be (re)constructed, fast to predict values, and efficient to send parameter values for model update from a producer node to a consumer node.

Chebyshev Regression.
Chebyshev polynomials are also often used for representing and processing time-series data, due to the capability of efficiently computing near-optimal approximations for given streams.

Suppose that time values $t \in s$ vary within a range $[\min(t), \max(t)]$. We then obtain normalized time values t' within a range $[-1, 1]$, by a transformation function

$$f(t) = (t - \frac{\max(t) + \min(t)}{2}) \cdot \frac{2}{\max(t) - \min(t)}.$$

Its reverse transformation function $f^-(t')$ is then defined as:

$$f^-(t') = (t' \cdot \frac{\max(t) - \min(t)}{2}) + \frac{\max(t) + \min(t)}{2}.$$

Given a degree d, Chebyshev polynomial is defined as:

$$v_t = f^-(\cos(d \cdot \cos^{-1}(f(t)))) \qquad t = 1, ..., n$$

Let v_t^d be a Chebyshev polynomial v_t of degree d. We present the first several Chebyshev polynomials with increasing degree:

$$\begin{aligned} v_t^0 &= 1 \\ v_t^1 &= t \\ v_t^2 &= f^-(2f^2(t) - 1) \\ v_t^3 &= f^-(4f^3(t) - 3f(t)) \\ v_t^4 &= f^-(8f^4(t) - 8f^2(t) + 1) \end{aligned}$$

With a linear combination of the above Chebyshev polynomials having different degrees, Chebyshev polynomials are redefined by a generic form:

$$v_t = \beta_0 \cdot v_t^0 + \cdots + \beta_m \cdot v_t^m, \qquad t = 1, ..., n$$

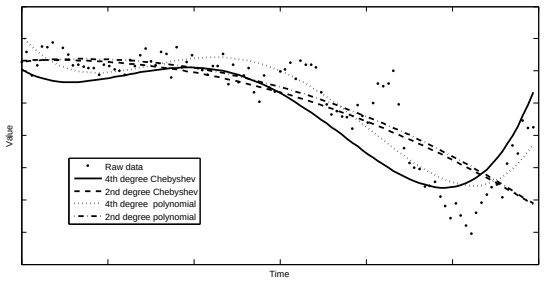

Fig. 2. Examples of Time-Series Regressions

Figure 2 demonstrates different applications of the regression models with varying degree over real data. As shown in the figure, polynomials with higher degrees can represent the raw stream with more detailed curves. Nevertheless, when the given data stream contains large amounts of noises, the curves captured by the high-degree polynomials may become fluctuated, which may lead to imprecise value prediction.

B. Adaptive Construction of Regression Model

Let $w = \langle v_g, v_{g+1}, \cdots, v_n \rangle, (1 \leq g < n)$ be a window (i.e., subsequence) of a stream $s = \langle v_1, v_2, \cdots, v_n \rangle$, where n is the time when the last sensor reading v_n was streamed. Let $|w|$ also be the size of the window w, i.e., the number of stream elements in w.

In general, regression models built by larger windows yield more precise value predictions than those built by small windows, because real sensor data typically contains errors or noise, and thus the trend of the stream over time is not effectively captured by the regression models using small windows. For example, when we use $|w| = 3$ for building a regression model in Fig. 2 (note that the curves in the figure are obtained by using all of the raw values), it is hard to expect that the model can predict next values precisely.

On the other hand, when we attempt to use large windows for constructing the regression model, it may violate the accuracy guarantee in the framework. For instance, suppose that a model-driven value v'_t generated by a regression model $\mathcal{M}_s$ using $|w| = 10$ exceeds a given error bound ϵ from its raw reading v_t (i.e., $|v_t - v'_t| > \epsilon$). We then reconstruct $\mathcal{M}_s$ using a smaller window size, e.g., $|w| = 9$. However, $|v_t - v'_t| > \epsilon$ may occur again although we use $|w| = 9$, because $\mathcal{M}_s$ derives its curves reflecting all the values in w. Therefore, we cannot determine an absolute window size for constructing a given regression model, yet $|w|$ needs to be computed dynamically.

To reflect the above discussions, we propose an algorithm that adaptively finds an appropriate window size. The key idea underlying in this algorithm is to find the maximum size of the window among the windows that can meet the accuracy guarantee. In order to do this, we consider a given default window size and then shrink or enlarge the size until we find an error-bounded largest window size.

Algorithm 1 presents the pseudo-code for (re)computing a given regression model dynamically. In Lines 1–3, variables

Algorithm 1 Adaptive Model Computation

Input: stream s, regression model $\mathcal{M}_s$, error bound ϵ_s, default window size $|w_o|$

```
1:  s = ∅
2:  window w ← ∅
3:  M_s ← ∅
4:  while s ← a new value v_t do
5:      w ← v_t
6:      if |w| ≥ |w_o| and M_s = ∅ then
7:          M_s ← construct using w
8:          send M_s to consumer nodes
9:      else
10:         continue
11:     model-driven value v'_t ← M_s(t)
12:     while |v_t − v'_t| > ε_s do
13:         M_s ← construct using w
14:         w ← remove the first (oldest) value
15:     if |w| ≠ |w_o| then
16:         send M_s to consumer nodes
```

are initialized. The algorithm then fills streaming values to the window w (Lines 4–5 and Lines 9–10), until the window has enough values, i.e., $|w| \geq |w_0|$ in Line 6. When this is done, the given regression model is computed, and the coefficients for the model are also sent to the consumer nodes who requested the stream s (Line 8). At Line 11, the model creates a model-driven stream v', in order to validate the accuracy guarantee of the framework (Line 12). When this condition is violated, the algorithm recomputes the model and shrinks the window size for the next iteration. This is repeated adaptively until the condition is satisfied (Lines 13–14).

This algorithm is generic, which any class of regression models can become an input of the algorithm, as well as it always guarantees the given accuracy.

IV. Coded Model Update

Although transmitting parameter values for models over networks is much more efficient than sending actual data streams, this benefit may decrease when the values in the stream fluctuate dramatically over short terms and thus model updates from one node to another occur often. To cope with this problem, we introduce a novel scheme that enables the model update more efficiently.

A. Overview

The core idea underlying our approach is to share a set of predetermined parameter values for building a model between a producer node and a consumer node. The framework then transfers merely bitmap-encoded identifers of the prearranged parameter values when a model update is required, instead of sending the actual parameter values for the model.

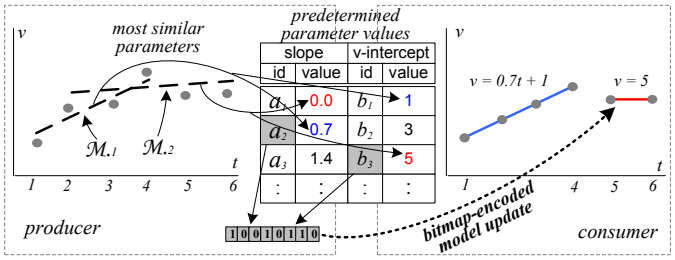

Fig. 3. An Example of Coded Model Update

Fig. 3 illustrates an example of the coded model update for a linear regression model. First, the producer node computes two sets of predetermined parameter values for the linear model (i.e., slopes and v-intercepts), such that each parameter value has a distinct identifier as a_i or b_i in the figure. It then sends the information of the predetermined values to the consumer node when the connection between the two nodes is initially established. Next, the information is subsequently updated only when necessary. In fact, we do not store the predetermined parameter values in the system, but derive them from their upper and lower bounds (the next subsection provides details).

After the initialization, the producer node represents the stream as a model instance $\mathcal{M}_{\cdot 1}$ during $t \in [1, 4]$, and it simultaneously matches the actual parameters for $\mathcal{M}_{\cdot 1}$ to the most similar preset parameter values (i.e., 0.7 and 1 for slope and v-intercept, respectively). While this, the producer node also monitors whether Property 1 holds between each actual sensor reading and its corresponding model-driven value obtained from the preset model. At the consumer node, the model-based view is generated by the predetermined model during the same time period $t \in [1, 4]$.

In this example, the actual reading at $t = 5$ at the producer node is assumed to exceed a given error bound from its corresponding model-driven value generated by the preset model, requiring the producer node to reconstruct the actual model as $\mathcal{M}_{\cdot 2}$. The producer node then finds the most similar preset parameter values (i.e., 0.0 and 5 in Fig. 3) to $\mathcal{M}_{\cdot 2}$

again. Next, it encodes a bitmap using the identifiers (i.e., a_2 and b_3) of the preset parameter values found and sends the bitmap to the consumer node where derives the parameter identifiers by decoding the bitmap. Since $t = 5$, the consumer node generates its model-based view by the model constructed by the predetermined parameter values having a_2 and b_3 identifiers.

The key features of this coded model update are fourfold:

- First, it permits the model update to be significantly more efficient because the necessary parameter information for model update is coded into a compact bitmap. In addition, this effect increases dramatically as the number of parameters required for models grows.
- Second, this method still supports the generic properties of the framework, such as ensuring accuracy guarantees and flexible application for any given models plugged into the framework.
- Third, its maintenance costs (i.e., storage requirement, model computation, and searching of preset parameter values) are very low.
- Lastly, its necessary system parameters are computed in automated manners without taking any inputs from user, while preserving all the above features.

B. Presetting Parameter Values

Let $P = \{p_i\}$ be a set of parameters required for building a given model, excluding constants (e.g., $P = \{\alpha_1, \alpha_2\}$ for a second-degree polynomial $v = \alpha_0 + \alpha_1 t + \alpha_2 t^2$). While sweeping a stream s, we obtain a set of model instances from the model's reconstructions when the difference between an actual reading and a model-driven value exceeds a given error bound. By extracting the value for p_i from each model instance, we collect a set V_{p_i} of parameter values. For example, $V_{p_1} = \{2, 4\}$ and $V_{p_2} = \{3, 5\}$ are obtained from two instances of degree-2 polynomials $v = 2t+3t^2$ and $v = 4t+5t^2$, respectively. Our framework then stores the upper and lower bounds of each V_{p_i}, denoted as $B_{p_i} = [\min(\bar{v}), \max(\bar{v})], \bar{v} \in V_{p_i}$. Similarly, it also stores those bounds of sensor readings in the stream, i.e., $B_v = [\min(v), \max(v)], v \in s$. Therefore, we maintain $|P|+1$ pairs of upper and lower bounds for parameter values and readings in the framework.

Given an integer number h, the space of B_{p_i} is conceptually divided into h subspaces, each of which has an equal size of $\frac{|B_{p_i}|}{h}$, where $|B_{p_i}| = \max(\bar{v}) - \min(\bar{v}), \bar{v} \in V_{p_i}$, e.g., $\langle [1, 3), [3, 5] \rangle$ for $h = 2, B_{p_i} = [1, 5]$. We then take the median value of each subspace to be used as a predetermined value for a parameter p_i. Note that we do not store these predetermined parameter values in the system but derive them from the bounds, reducing storage requirement for the coded model update substantially.

Let $floor(x)$ be a function that returns the largest integer value that is not greater than x. Given an actual parameter value v_p, its nearest predetermined parameter value is computed by

$$\min(\bar{v}) + h \cdot (floor(\frac{v_p}{h}) + \frac{1}{2}) \qquad \bar{v} \in V_{p_i}, v_p \in B_{p_i} \quad (2)$$

Equation 2 has a constant-time complexity. Therefore, all of the necessary parameter information for building the model can also be computed with a constant-time complexity, i.e., $O(|P| + 1)$.

As time passes and s receives more new readings, the space of $|B_{p_i}|$ may be expanded, consequently each preset parameter value may also cover a large subspace. Nevertheless, such a expansion seldom occurs after certain time periods (e.g., the highest and the lowest values of air temperature over years do not change often). In addition, the large space of $|B_{p_i}|$ does not necessarily mean that the preset parameter values have coarse granularities. For instance, coefficient values associated with the time variable of polynomial regression curves or lines can be normalized within $[-\frac{\pi}{2}, \frac{\pi}{2}]$. As a more specific example taking a linear regression model and $h = 10$ (which is much smaller than its typical values in our system), the angle between the line formed by the actual model and that formed by the model using the preset parameter values always stays within at most an 18-degree error, regardless of $|B_{p_i}|$. Furthermore, even if the model constructed by the preset parameter values may require more frequent model update due to its inaccurate prediction compared to the actual model, each size of model update is still much smaller than that of actual parameter values, rendering lower total costs for model update.

C. Error Bounding

Let v' and v^* be model-driven values, generated from an actual model $\mathcal{M}$ and a model $\mathcal{M}^\circ$ constructed by a set of preset parameter values that are most similar to the parameter values for $\mathcal{M}$, respectively. Given a given error bound ϵ and a raw reading value v, it may occur that $|v - v'| \leq \epsilon$ but $|v - v^*| > \epsilon$ (Fig. 4(a)), which violates the accuracy guarantee in our framework and thus $\overline{\mathcal{M}}$ is cannot be directly used for model update.

To address this, we first ensure that each subspace size in B_v is not greater than the double size of the accuracy bound, i.e., $\frac{|B_v|}{h} \leq 2\epsilon$, when we predetermine the reading values in B_v. Therefore, the closet preset reading value c to v must not exceed ϵ from v. Next, we compose a $\mathcal{M}^*$ combining $\overline{\mathcal{M}}$ and c as a constant, such that $\mathcal{M}^*(t) = \overline{\mathcal{M}}(t)+c$, where $\overline{\mathcal{M}}(0) = 0$. After updating this $\mathcal{M}^*(t)$ to consumer nodes, model-driven values are generated by $\mathcal{M}^*(t), t = 0, 1, \cdots$ at both producer and consumer nodes. Recall that we removed any constants for $P = \{p_i\}$ in subsection IV-B, thus $|v_0 - \mathcal{M}^*(0)| \leq \epsilon$ always holds (because $\mathcal{M}^*(0) = c$). By repeating the above processes whenever $|v_t - \mathcal{M}^*(t)| > \epsilon, t > 0$, the accuracy is always guaranteed.

Fig. 4(b) demonstrates an example of the procedure for the accuracy guaranteed model update using linear regression model. During $t = [1, 3]$, model-driven values are created by a model instance $\mathcal{M}.^*_1$ built by preset parameter values. At $t = 3$, $|v - v^*| > \epsilon$, requiring a model update. We then build an actual mode $\mathcal{M}$ over the past values of the stream, and find the closest preset reading value c to v, as well as the most similar preset value of slope the slope of M. Next, $\mathcal{M}.^*_2$ is constructed by the preset values found. Since $t = 3$, we count the time

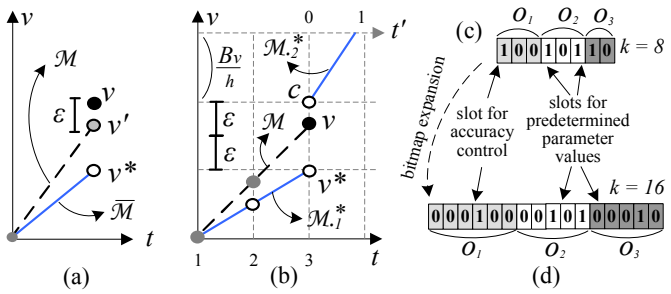

Fig. 4. Accuracy Guarantees for Coded Model Update

in a different way as $t' = 0, 1, \cdots$ and generate model-driven streams by $\mathcal{M}^*(t')$ at both producer and consumer nodes.

D. Bitmap Encoding

We consider k bits as the size of a bitmap for the message of coded model update and initially set $k = 8$. The k bits are then distributed into a sequence of slots $\langle o_1, o_2, \cdots, o_{|P|+1} \rangle$, where o_i has an equal or similar number of bits to another (Fig. 4(c)). Specifically, we first assign $floor(\frac{k}{|P|+1})$ bits to each o_i. We then assign one more bit to each slot for $(k - floor(\frac{k}{|P|+1}) \cdot (|P| + 1))$ times, from o_1 along the sequence order. Given $k = 8$ and $|P| = 2$, for example, we obtain $|o_1| = 3, |o_2| = 3$, and $|o_3| = 2$, where $|o_i|$ is the number of bits assigned to o_i slot. Each slot in the bitmap is easily identified by applying a mask for the bitmap.

The first slot o_1 is reserved to encode a predetermined reading value in B_v, and each of the rest slots is designed to encode a predetermined parameter value in B_{p_i}. Thus, o_1 can represent $2^{|o_1|}$ distinct numbers from 0 to $2^{|o_1|}$-1, which are the identifiers for predetermined readings. Likewise, each $o_j (j \neq 1)$ can describe $2^{|o_j|}$ different predetermined parameter values.

When the number of parameters $|P|$ required for building a model is large, the initial setting of $k = 8$ may be insufficient for describing the details of each parameter for the model well. For such a case, we incrementally append 8 more bits to the last value of k as Fig. 4(d) until both of the following conditions are satisfied: First, the preset parameter values for readings derived from $|B_v|$ meet the accuracy guarantee, i.e., $\frac{|B_v|}{|o_1|} \leq 2\epsilon$. Second, the granularity of the predetermined parameters should not be too coarse, i.e., $\min(|o_j|) < 4, (j \neq 1)$, thus the example of Fig. 4(c) does not satisfy this condition. When we increase the k value, we fill up more bits to the slots where lower numbers of bits were assigned for the distribution of the k bits at the previous steps, so that each $|o_j|$ is nearly the same as that of another slot.

V. Coded Inter–Variable Model

It is often observed that data streams collected from different observations or locations exhibit correlated trends over time due to physical laws. For instance, Fig. 5(a) plots sensor readings of air temperatures from two different places, and Fig. 5(b) does measurements of air and ground temperatures in the same area. Such correlated streams generally compensate errors and noises by exploiting the dependencies from one stream to another. This can yield more precise value prediction and thus model updates from producer nodes to consumer nodes can also occur less frequently.

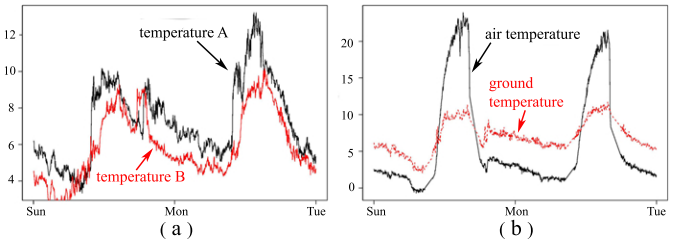

Fig. 5. Examples of Correlated Streams over Real Data

A. Preliminaries

In Section III, polynomial and Chebyshev regression models are used to describe how a *physical variable* (i.e., sensor reading) depends on another *temporal variable* (i.e., time). Such a dependency may also be found between two different physical variables. In this section, we exploit the correlations across the variables, and utilize them in our framework when the correlated streams are requested from consumer nodes. Since our coded model update is designed to support any given arbitrary models, we aim to develop this correlation model based on the coded model update method. Thus, it is natural to expect synergy effects from combining the coded model update and the correlation models.

Specifically, we consider a base stream that is represented by a (preset) model $\mathcal{M}^*_{s_i}$. Next, we take into account linear dependency with scale factor (i.e., a constant) from $\mathcal{M}^*_{s_i}$ to another model $\mathcal{M}^*_{s_j}$ for a stream s_j. By doing this, the trend of the model-based view obtained by $\mathcal{M}^*_{s_i}$ shows similar behaviors to that by $\mathcal{M}^*_{s_j}$, which reflects the correlations over original streams s_i and s_j. Due to the scale factor, streams having different absolute values can form correlations as long as the streams show similar trends over time. We formally define this model as follows:

Definition 2: Given two streams s_i and s_j, a scale factor δ, and the most similar predetermined model $\mathcal{M}^*_{s_i}$ to its corresponding actual model for s_j, a **coded inter–variable model** $\mathcal{M}^\circ_{s_j}$ for s_j is defined as a function of $\mathcal{M}^*_{s_i}$:

$$\mathcal{M}^\circ_{s_j} = \delta \cdot \mathcal{M}^*_{s_i}$$

It has been shown that using linear dependency with scale fact across variables is very effective to handle correlated streams, in terms of minimizing data redundancy [19], [20].

Fig. 6(a) illustrates an example of how the *coded inter-variable model* works, using piecewise constant models for representing two streams s_1 and s_2 that are registered to the framework as correlated streams. $\mathcal{M}_{s_i \cdot j}$ and $\mathcal{M}^*_{s_i \cdot j}$ denote the j-th instances of actual model $\mathcal{M}_{s_i}$ (i.e., base model) and its most similar predetermined model $\mathcal{M}^*_{s_i}$, respectively. $\mathcal{M}^\circ_{s_1 \cdot j}$

is the j-th instance of the coded inter-variable model having $\delta = 3$ that is shared by both producer and consumer nodes.

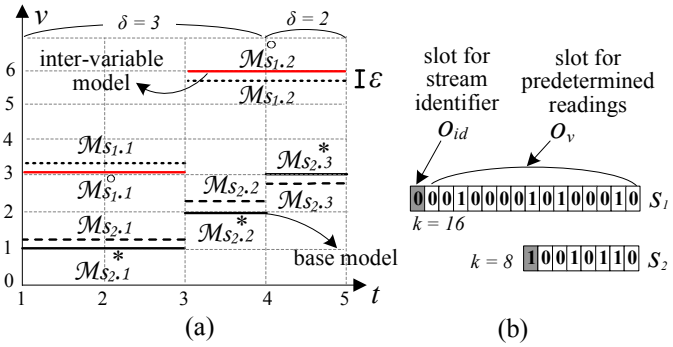

Fig. 6. An Example of Coded Inter-Variable Model

At $t = 3$, both $\mathcal{M}^{\circ}_{s_1}$ and $\mathcal{M}^{*}_{s_2}$ change, as well as the corresponding actual models $\mathcal{M}_{s_1}$ and $\mathcal{M}_{s_2}$ vary. In this case, the framework does not update the parameter value for $\mathcal{M}^{\circ}_{s_1 \cdot 2}$ but only for $\mathcal{M}^{*}_{s_2 \cdot 2}$, because the initial values $\delta = 3$ (i.e., $\frac{\mathcal{M}^{\circ}_{s_1 \cdot 2}}{\mathcal{M}^{*}_{s_2 \cdot 2}}$) is not changed. Thus the value for $\mathcal{M}^{\circ}_{s_1 \cdot 2}$ is computed by Definition 2 at the consumer node where keeps the value of δ. This reduces the sizes of update messages. Especially when the number of correlated streams is large, this effect increases.

At $t = 4$, only $\mathcal{M}^{*}_{s_2 \cdot 2}$ is changed to $\mathcal{M}^{*}_{s_2 \cdot 3}$. For this case, our framework sends the parameters for both $\mathcal{M}^{\circ}_{s_1 \cdot 2}$ and $\mathcal{M}^{*}_{s_2 \cdot 3}$, which are 3 and 6 respectively. Then, the consumer node recomputes the values for δ, so that the new value of the scale factor (i.e $\delta = \frac{6}{3}$) is used for describing the dependency between the two models.

B. Bounding Errors and Presetting Parameters

One of key challenges for the development of the coded inter-variable model lies in supporting the accuracy guarantee in our framework. Although the coded model update adjusts the subspace sizes of upper and lower bounds, i.e., B_v in reading values (see Section IV-C), model-driven values using an inter-variable model are derived by a base model, thus the accuracy guarantee method in the coded model update is not directly applicable.

Recall the example of the model updates at $t = 4$ in Fig. 6(a). We did not send a new δ value but it was computed at the consumer node, although δ is a parameter of the coded inter-variable model. This is a simple but efficient manner to bound a user given error. In fact, the size of update message for a δ value and that for the closet predetermined value to the current raw reading are identical, and the δ value can be derived from the predetermined value. If we maintain the upper and lower bounds B_v for the predetermined values, we can apply the same method as that the coded model update bounds errors in Section IV-C for guaranteeing the accuracy in the coded inter-variable model.

Reflecting the above ideas, we prearrange parameters as follows. Given a set of piecewise constant models corresponding to a set correlated streams, we take one model $\mathcal{M}^{*}_{s_b}$ as a base model, and the others $\mathcal{M}^{\circ}_{s_i}$ for inter-variable models. For each model, we maintain a pair of upper and lower bounds, i.e., B_v. We then predetermine parameters for each B_v, following the same manners described in Section IV-C.

After prearranging the parameter values, both producer and consumer nodes share the predetermined parameters. In addition, they also share the information of model classes that are categorized into three cases; base model, inter-variable model, and non-correlated model. For inter-variable models, initial δ values are also stored at both sides. Therefore, when a producer node updates a model with an identifier associated with its corresponding stream, the consumer node takes different model updates according to the model class distinguished by the identifier.

C. Bitmap Encoding

Suppose that a set of streams $S = \{s_1, s_2, \cdots, s_n\}$ at a producer node are requested from a consumer node. Starting with $k = 8$ bits for a bitmap as an update message in the coded model update, we divide the k bits into two slots, which the first slot o_{id} is assigned for identifiers of each stream and the other o_v is designed for the accuracy control slot. We fist assign $|o_{id}| = ceil(\log_2(n))$ within k, where $ceil(x)$ returns the smallest integer value that is not smaller than x, and the rest bits of k are assigned to o_v(e.g., Fig. 6(b)).

Given an error bound ϵ_{s_i} for a stream s_i, whenever $\frac{|B_v|}{2^{|o_v|}} \leq 2\epsilon_{s_i}$ (o_v, B_v for s_i) does not hold, we expand the size of the bitmap as we do for the coded model update (see Section IV-C). This is to guarantee the accuracy described in the previous subsection. Note that $|o_{id}|$ is identical for the bitmaps associated with $s_i \in S$, however, each $|o_v|$ for s_i may be different from another because each stream may have different sizes of upper and lower bounds $|B_v|$.

The coded inter-variable model can assign more bits to the accuracy slot o_v than the general coded model update does (e.g., Fig. 6(b)), because o_v generally does not take many bits of k unless the number of streams requested by a consumer node is very large. As a result, the preset models become more descriptive by having more preset readings, rendering more precise value prediction.

D. Computing Correlated Streams

A recent study [20] in stream data compression presents a state-of-the-art solution, called *GAMPS*, that elaborates many methods such as finding optimal groups of streams for applying correlation models, and computing optimal base streams with respect to maximized data compression. Unfortunately, some of these methods cannot be directly employed for our work, because they consider compression of static historical data that are already collected, while our work applies to real-time data. In this study, we aim to minimize the size of data and computational cost for identifying a specific stream, to be sent from producer nodes to consumer nodes. When the number of streams requested by a consumer node is large, discovering which stream is correlated to another or others is computationally expensive. In addition, sending the identifiers

of correlated streams may need an even larger data size than the size of parameter values for model update.

To cope with this, we consider a given time-window that contains some histories of the data streams requested by a consumer node. We then discover correlated streams within this window, and then apply our coded inter-variable model for the correlated streams found. The intuition behind this is that correlated streams in the window are also likely to exhibit similar trends for longer terms. Thus, we do not need to compute them every time when a producer node receives a new reading. For example, the streams in Fig 5 show similar trends other over three days. Those plots suggest that we can detect such correlations with small window sizes, e.g., half a day.

Given a window and a set of streams to be transmitted to a consumer node, we discover groups of correlated streams by utilizing the plane-sweep algorithm. Our cost model is the frequency of model updates while scanning the window along time. We compute the cost for every combination of the streams. If the combination contains only one stream, we use a given model, otherwise the coded inter-variable is applied to obtain the cost for this combination. Next, we select the combinations having the lowest costs as the groups of correlated streams. For each of such groups, we apply the method introduced in GAMPS [20] for finding the base signal for this group. Note that this operation is performed only once when a consumer node requests multiple streams that are maintained by a producer node.

VI. Experiments

The objective of our experimental study in this section is threefold. First, we empirically demonstrate that using model-based views significantly improves the efficiency of data communication over networks. We present the communication costs of various models with varying degree. Second, we analyze the effect of our coded model update (Section IV) in terms of communication efficiency and model maintenance cost. Last, we compare the performance of our coded inter-variable model (Section V) with a state-of-the-art solution (GAMPS [20]).

We implemented the above methods in the JAVA language and MATLAB on a Windows XP operating system. All the methods ran on an Intel Core 2 Duo processor 2 GHz system with 2 GB of main memory.

A. Datasets and Cost Measures

Our experiments use two real datasets in order to contend with real phenomena, each of which contains an entire class of sensor deployment for environmental monitoring. Details for each dataset are described below, and Table II summarizes their key features:

- **St. Bernard**: A large collection of data was being obtained for a period of 7 months at the Grand St. Bernard mountainous area in Switzerland. We randomly chose one deployment station and obtained 8 distinct data streams that measured either same or different observations for a week. The observations are air temperature, surface temperature, relative humidity, solar radiation, soil moisture, and wind speed. Some of the streams in this dataset show quite similar trends over time, i.e., correlations.
- **Wannengrat**: Six different observations were recorded over a long period of time in Wannengrat, Switzerland. We obtained some of them, measured for two months. The observations include relative humidity, air temperature, surface temperature, snow height, wind speed, and wind direction. Unlike St. Bernard, correlations of the streams are hardly found in this dataset.

name of dataset	**St. Bernard**	**Wannengrat**
number of streams	8	6
number of spatial locations	6	1
number of physical variables	3	6
duration of measurements	1 week	2 months
sampling rates	1 sample/2min	1 sample/10min
average number of readings	10080/stream	17448/stream
total number of readings	80640	104688

TABLE II
Summary of Datasets

Throughout this experiment section, we present comparisons of communication costs with different applications of models and error bounds. We computed the communication costs as follows. Let $|s|$ be the number of readings in an original data stream s. Let $|s'|$ also be the number of model updates occurred with respect to a given error bound, while sweeping s from its beginning to the end. We then obtain a communication cost for the stream s by $cost_s = \frac{|s'|\cdot|u|}{|s|\cdot 4bytes}$, where $|u|$ is the size (byte) of one model update, and $4bytes$ denotes that each reading has a 4-byte data size. We next compute the cost for every stream in a dataset, and obtain the total cost by aggregating them. Therefore, $cost_s = 0.1$ means that using model-based views takes a ten percent of the original data size to be transmitted from one producer to another consumer node.

To specify user-given error bounds for experiments, we first compute the difference between the maximum and the minimum values in a stream. We then use a certain proportion to the difference for the value of the error bound. For instance, suppose that a stream has 1000 and 1 as the maximum and the minimum values, respectively. Error bound $= 1\%$ indicates that the size of the error bound is 10.

B. Effect of Coded Model Update

In the first set of experiments, we present the effect of using model-based views for minimizing communication cost over networks. Fig. 7 clearly demonstrates significant improvements of communication efficiency, when we utilize the models for data communication over networks. To obtain these results, we set the user-given error bound to 1 %. In the results, our proposal achieves at best over 99 % (coded 0-degree polynomial regression for St. Bernard) and at least 91 % (2-degree Chebyshev regression for St. Bernard) less data communications, compared to transmitting original data.

Comparing time-series models with coded time-series models, the power of our coded model update is obviously shown in any case of the experiments. This is natural since the coded model update compresses the model parameter values with compact bitmaps. For these experiments, the sizes of the bitmaps (i.e., k in Section IV-D) were only one or two bytes for each stream. Thus, using the coded model update for a model having one parameter should have theoretically at least a twice smaller communication cost than without coding, however, this assumption does not hold in some cases in these experiment sets. Because the coded model update uses predetermined parameter values which may not be sufficiently descriptive, it causes more frequent model updates although each update has a substantially smaller amount of data.

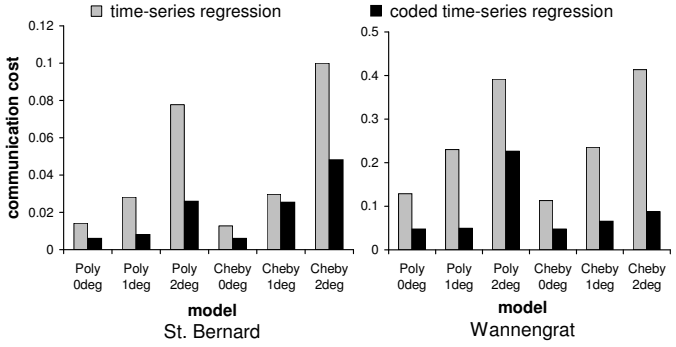

Fig. 7. Comparison of Communication Cost

For time-series models, lower degree polynomials show better communication efficiency than do higher degree ones for both polynomial and Chebyshev regressions. This is because higher degree regressions take larger numbers of parameters to be transferred over networks. For instance, a polynomial regression with degree 0 (i.e., piecewise constant) has only one parameter for model update, while that with degree 2 has three parameters, incurring three times higher model-update costs. In theory, high-degree regressions should achieve better approximation results for streams, thus they should also yield more precise value predictions, leading to infrequent model updates. In practice, however, sensor data is generally noisy and erroneous, decreasing the theoretical advantages for the high-degree regression models.

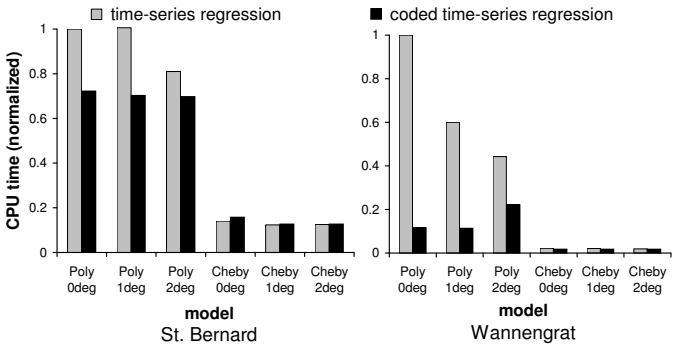

Fig. 8. Comparison of Model-Maintenance Cost

Next, we analyze the model maintenance costs at producer-node side, which are reconstructing models while guaranteeing accuracy, and generating model-driven values in our framework. Fig. 8 compares these under the same settings as the previous experiments. We measure the costs as total computing (CPU) time of simulating the model updates, while scanning the whole streams from their beginnings for each data. We normalize the values resulted by setting the longest elapsed time to 1.0.

Among the results from time-series models, Chebyshev regressions take substantially smaller amounts of computing costs, because the characteristics of Chebyshev approximation (Section III-A). Among the results from polynomial regressions, we find a very interesting fact. As polynomials have higher degrees, their computing costs decrease for both datasets, although high-degree polynomial regressions have more expensive costs for model construction. The reason is to guarantee accuracy. In Section III-B, Algorithm 1 takes a default window size as an input and incrementally finds a 'good' final window size until errors are bounded. In our experiments, we find out that polynomials having low degrees generally involve more numbers of the increments to determine the final window.

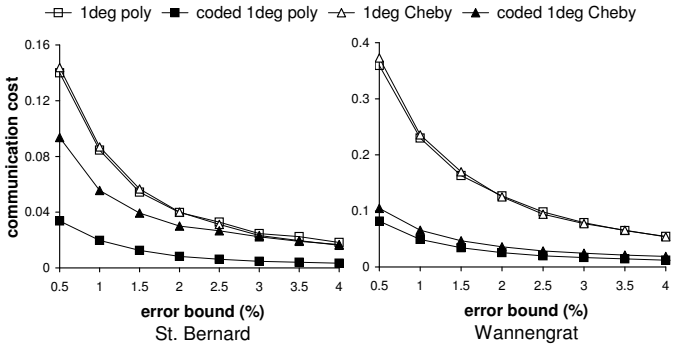

Fig. 9. Effect of Error Bound

In the following experiment, we study how varying error bound affects the performance of our framework. Fig. 9 shows the changes of communication cost along different sizes of error bound for first degree polynomial and Chebyshev regression models with or without our coding. As expected, larger sizes of error bound increase the efficiency of data communication, because models are not updated unless any model-driven value exceeds the error bound from the corresponding raw reading, described in Property 1 in Section II.

This observation becomes more clear when the error bound grows from 0.5 % to 2 % and then it does less after 2 %. Using a such value of error bound as 2 %, called a *knee point*, implies that our framework shows a great performance, in terms of minimizing the size of error bound and maximizing communication efficiency. If an application using the framework needs to set the value for error bound in an automated way, it is ideal to use such a knee point for the value.

Another observation found in Fig. 9 is that coded models are less sensitive to the changes of error bound for the com-

munications. This also implies that the knee points may exist even before the value 0.5 % of error bound for these cases, thus the coded model update with a setting of a much smaller error bound may show a similar performance to general time series models using higher values for the error bound.

C. Effect of Coded Inter-Variable Model

We next compare the communication costs of our coded inter-variable model with a state-of-the-art solution GAMPS [20]. Both are correlation models and share similar underlying ideas that exploit linear dependencies among the streams. Fig. 10 demonstrates their communication costs with varying error bound, as well as those for using piecewise constant models for the streams without taking into account their correlations.

For the St. Bernard dataset, our coded inter-variable model outperforms all the other competitors, and this becomes more remarkable for the Wannengrat dataset. GAMPS also performs better than the piecewise constant model for St. Bernard. This supports the discussion made in Section V-A, such that considering correlations of streams can reduce the redundancy of data, rendering more efficient data communication over networks.

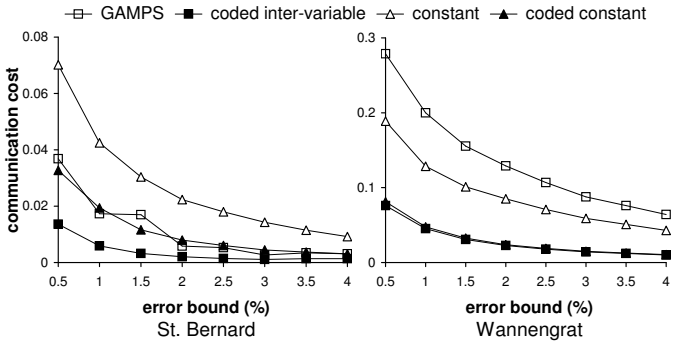

Fig. 10. Effects of Correlation Models

As we described in Section VI-A, the Wannengrat dataset seldom contains correlated streams. As a result, the communication costs using GAMPS increase substantially in this dataset. In contrast, our coded inter-variable model still shows the best performance among all the methods, and the differences of communication costs over the two datasets are low for our method. This is because the coded inter-variable model works like the piecewise constant model, when streams are uncorrelated. Thus, these experiments suggest that our coded inter-variable model is effective and efficient to deal with not only correlated streams, but also uncorrelated streams.

Continuing the experiments in Fig. 10, our method exhibits at least approximately twice better efficiencies than does GAMP in most cases. There are two key reasons for this. First, our model is based on the coded model update, reducing the size of data to be transmitted over networks. Second, because this study considers only real-time data processing, full functionalities provided by GAMPS could not be adopted for these tests, described in Section V-D, since GAMPS is designed for static historical datasets.

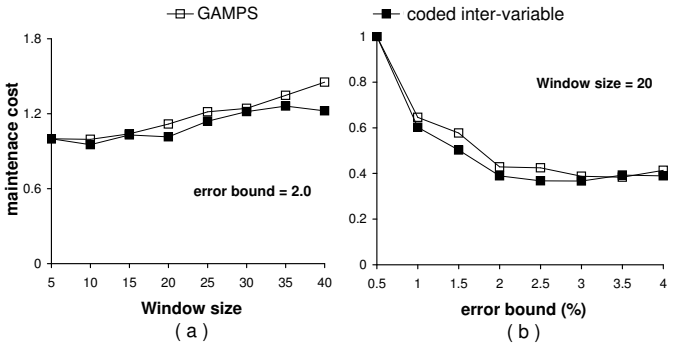

Fig. 11. Model-Maintenance Cost of Correlation Models

Lastly, we examine how the model maintenance costs of both methods change. Fig. 11(a) demonstrates the changes over increasing window size (i.e., the default window size as an input for Algorithm 1), and Fig. 11(b) does over varying error bound. These results are obtained from the St. Bernard dataset. We omit the results obtained from the Wannengrat dataset due to similar results. The values in the graphs are normalized to the first ones.

In these two experiment sets, both GAMPS and coded inter-variable demonstrate very similar trends of the results because they have the same underlying model, i.e., piecewise constant. Notice that, however, our model outperformed GAMPS in the previous experiments, with these similar computational costs.

In Fig. 11(a), the two methods show slight increases of the maintenance costs along increasing window size. In contrast, they exhibit clear decreasing of the costs with increasing error bound in Fig. 11(b). The reason is that the number of model reconstructions decreases dramatically when the error bound grows, as discussed in Fig. 9.

VII. Related Work

Due to the characteristics of continuity, data streams are often modeled by continuous-time functions as *time-series regression models* [15], [10], [16]. The main focuses of these studies are, however, not on minimizing communication cost over distributed network settings, but on developing techniques for query processing in centralized system settings.

Probabilistic models [11], [12], [13] and *PCA* (Piecewise Constant Approximations) [18] are employed for energy-efficient data communications. While their methods are designed for specific models in local sensor networks, our framework targets arbitrary models in federated sensor networks.

Instead of building individual models for single data streams, *correlation models* for multiple streams have also been highlighted, particularly for data stream compression [19], [23], [20]. They can generally increase compression ratios by reducing data redundancy. Although our proposal also takes into account stream correlation, it differs from them because they mainly consider static historical data, whereas our work applies to real-time data.

Streaming data dissemination [3], [4], [5] concerns continuous data transfers from producer nodes to consumer nodes. These studies assume that raw data streams must be disseminated. Hence, they focus on maximizing the shares of data to be carried together over networks. In contrast, we claim that conveying the raw streams is unnecessary; we transfer only the models rather than the actual streams, resulting in significant reductions of the data communication cost.

In contrast to moving the actual data streams, *placing operators* (or executable codes) into networks has also been studied in a rich body of research work [6], [7], [8], [9]. The key disadvantage of these work, however, is that query results must be delivered across networks, even if the results are optimally reorganized with respect to network latency, maximal share for multiple queries, and so on. This may decrease the communication efficiency when the query results contain large amounts of data (e.g., SELECT *).

Monitoring distributed data streams [24], [25] also considers communication efficiency, while dealing with continuous processing of queries triggered for detecting some conditions. In our framework, an event satisfying such a triggered condition is easily detected over model-based views, without contacting each data source of the streams.

Optimizing query plans [26], [27] with minimum communication is another related area to this work. While the studies focus on specific query types, we aim to provide a generic framework that supports most types of queries with respect to data streams.

VIII. Conclusions

Increasing use of sensor network is resulting in federated sensor networks, consisting of interconnected local sensor networks. To reduce data communication in such a network, various proposals have been introduced. However, they are generally query-dependent or inefficient for large volumes of query results. This paper proposes a novel and generic framework that represents data streams by given arbitrary numerical models, so-called *model-based views*. Only the models' parameters are transferred over the networks for efficient data communication. Moreover, we propose a novel method that boosts the performance of the framework, named *coded model update*. It compresses even the parameter values of the models to be transmitted, by encoding them with compact bitmaps. We also present our *coded inter-variable model* that incorporates an effective correlation model into the efficient coded model update. Extensive experimental results using real data suggest that our proposals are highly communication-efficient.

References

[1] N. Dawes, K. A. Kumar, S. Michel, K. Aberer, and M. Lehning, "Sensor metadata management and its application in collaborative environmental research," in *eScience*, 2008, pp. 143–15.

[2] S. Michel, A. Salehi, L. Luo, N. Dawes, K. Aberer, G. Barrenetxea, M. Bavay, A. Kansal, K. A. Kumar, S. Nath, M. Parlange, S. Tansley, C. van Ingen, F. Zhao, and Y. Zhou, "Environmental monitoring 2.0," in *ICDE*, to appear, 2009.

[3] S. Shah, S. Dharmarajan, and K. Ramamritham, "An efficient and resilient approach to filtering and disseminating streaming data," in *VLDB*, 2003, pp. 57–68.

[4] S. Shah, K. Ramamritham, and P. Shenoy, "Maintaining coherency of dynamic data in cooperating repositories," in *VLDB*, 2002, pp. 526–537.

[5] Y. Zhou, B. C. Ooi, and K.-L. Tan, "Disseminating streaming data in a dynamic environment: an adaptive and cost-based approach," *The VLDB Journal*, vol. 17, no. 6, pp. 1465–1483, 2008.

[6] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The design of the borealis stream processing engine," in *CIDR*, 2005, pp. 277–289.

[7] M. Balazinska, H. Balakrishnan, and M. Stonebraker, "Load management and high availability in the medusa distributed stream processing system," in *SIGMOD*, 2004, pp. 929–930.

[8] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. Manku, C. Olston, J. Rosenstein, and R. Varma, "Query processing, resource management, and approximation in a data stream management system," in *CIDR*, 2003, pp. 245–256.

[9] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer, "Network-aware operator placement for stream-processing systems," in *ICDE*, 2006, p. 49.

[10] A. Deshpande and S. Madden, "MauveDB: supporting model-based user views in database systems," in *SIGMOD*, 2006, pp. 73–84.

[11] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive stream resource management using kalman filters," in *SIGMOD*, 2004, pp. 11–22.

[12] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong, "Approximate data collection in sensor networks using probabilistic models," in *ICDE*, 2006, p. 48.

[13] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *VLDB*, 2004, pp. 588–599.

[14] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, "Irisnet: An architecture for a worldwide sensor web," *IEEE Pervasive Computing*, vol. 2, no. 4, pp. 22–33, 2003.

[15] Y. Ahmad, O. Papaemmanouil, U. Çetintemel, and J. Rogers, "Simultaneous equation systems for query processing on continuous-time data streams," in *ICDE*, 2008, pp. 666–675.

[16] A. Thiagarajan and S. Madden, "Querying continuous functions in a database system," in *SIGMOD*, 2008, pp. 791–804.

[17] Y. Cai and R. Ng, "Indexing spatio-temporal trajectories with Chebyshev polynomials," in *SIGMOD*, 2004, pp. 599–610.

[18] I. Lazaridis and S. Mehrotra, "Capturing sensor-generated time series with quality guarantees," in *ICDE*, 2003, pp. 429–440.

[19] A. Deligiannakis, Y. Kotidis, and N. Roussopoulos, "Compressing historical information in sensor networks," in *SIGMOD*, 2004, pp. 527–538.

[20] S. Gandhi, S. Nath, S. Suri, and J. Liu, "GAMPS: compressing multi sensor data by grouping and amplitude scaling," in *SIGMOD*, 2009, p. to appear.

[21] C. Ré, J. Letchner, M. Balazinksa, and D. Suciu, "Event queries on correlated probabilistic streams," in *SIGMOD*, 2008, pp. 715–728.

[22] B. Kanagal and A. Deshpande, "Online filtering, smoothing and probabilistic modeling of streaming data," in *ICDE*, 2008, pp. 1160–1169.

[23] H. Chen, J. Li, and P. Mohapatra, "RACE: time series compression with rate adaptivity and error bound for sensor networks," 2004, pp. 124–133.

[24] R. Keralapura, G. Cormode, and J. Ramamirtham, "Communication-efficient distributed monitoring of thresholded counts," in *SIGMOD*, 2006, pp. 289–300.

[25] I. Sharfman, A. Schuster, and D. Keren, "A geometric approach to monitoring threshold functions over distributed data streams," *TODS*, vol. 32, no. 4, p. 23, 2007.

[26] J. Li, A. Deshpande, and S. Khuller, "Minimizing communication cost in distributed multi-query processing," in *ICDE*, 2009, p. to appear.

[27] A. Silberstein and J. Yang, "Many-to-many aggregation for sensor networks," in *ICDE*, 2007, pp. 986–995.