This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE Globecom 2011 proceedings.

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

Alexandru Arion, Hoyoung Jeung, Karl Aberer

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

***Abstract*—Minimizing communication cost is a fundamental problem in large-scale federated sensor networks. Maintaining model-based views of data streams has been highlighted because it permits efficient data communication by transmitting parameter values of models, instead of original data streams. We propose a 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, while 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 correlation model, named *coded inter-variable model*, that merges the efficiency of the coded model update with that of correlation models. Empirical studies with real data demonstrate that our proposal achieves substantial amounts of communication reduction, outperforming state-of-the art methods.**

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.

A rich body of research deals with this problem, including communication-efficient data dissemination [3], [4] and effective operator placement in distributed stream processing systems [5], [6], [7]. Although these proposals reduce large amounts of data transfers over distributed sensor networks, they are 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* [8], 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.
- Third, our framework is generic and any type of model can be employed, as long as the model is time-parameterized. This is important because a number of models are being used for different purposes in a wide range of applications.
- 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 [9], [10], [11] 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.

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

978-1-4244-9268-8/11/$26.00 ©2011 IEEE

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

II. The Framework

In this section, we introduce our framework that utilizes model-based views for efficient data communication over federated sensor networks.

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.

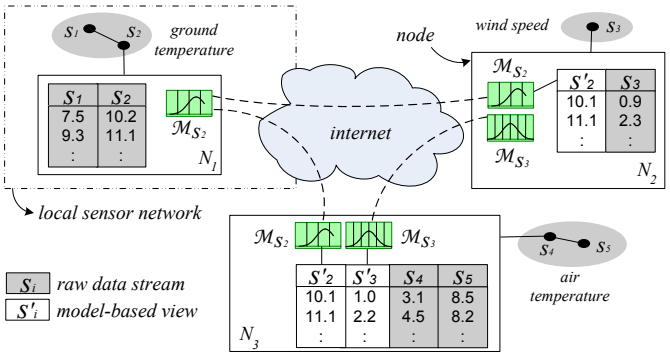

Fig. 1. An Example of The Framework over A Federated Sensor Network

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.

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, where a model-based view s'_2 for representing s_2 is generated. 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.

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*, is smaller than 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.

III. 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 identifiers of the prearranged parameter values when a model update is required, instead of sending the actual parameter values for the model.

Fig. 2 illustrates an example of the coded model update for a linear regression model. First, the producer node computes two

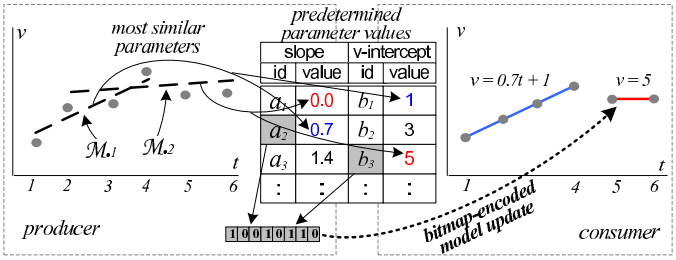

Fig. 2. An Example of Coded Model Update

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. 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 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. 2) 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 which derives the parameter identifiers by decoding the bitmap.

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. 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 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

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

Equation 1 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.

IV. Coded Inter–Variable Model

It is often observed that data streams collected from different sensors or locations exhibit correlated trends over time due to physical laws. For instance, Fig. 3(a) plots sensor readings of air temperatures from two different places, and Fig. 3(b) shows measurements of air and ground temperatures in the same area. Such correlated streams generally can be used to obtain more precise value predictions and thus model updates from producer nodes to consumer nodes can also occur less frequently.

A. Preliminaries

We utilize the correlations across physical variables in our framework when 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.

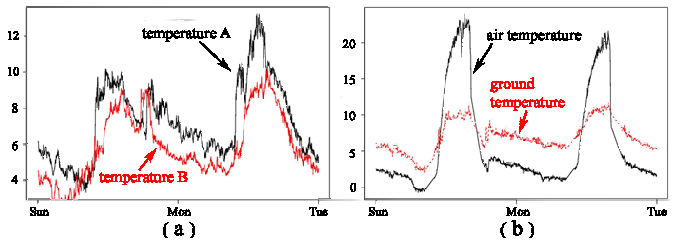

Fig. 3. Examples of Correlated Streams over Real Data

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_i, 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 factor across variables is very effective to handle correlated streams, in terms of minimizing data redundancy [12], [13].

Fig. 4(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.

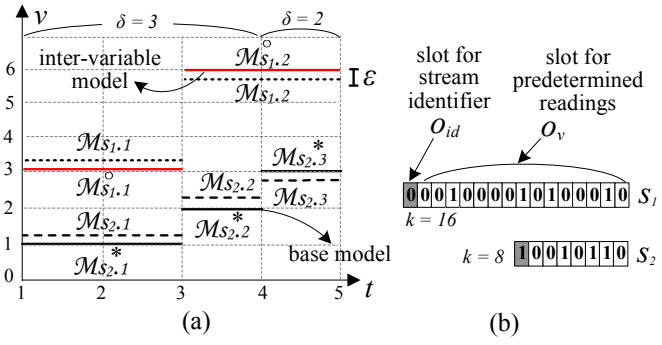

Fig. 4. An Example of Coded Inter-Variable Model

At $t = 3$, both $\mathcal{M}^\circ_{s_1}$ and $\mathcal{M}^*_{s_2}$ change, and the corresponding actual models $\mathcal{M}_{s_1}$ and $\mathcal{M}_{s_2}$ also 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. 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. 4(b)).

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. 4(b)), because o_id 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.

C. Computing Correlated Streams

A study in stream data compression [13] presents a state-of-the-art solution, called *GAMPS*, that finds optimal groups of streams for applying correlation models, and computes 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 costs 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.

To cope with this, we consider a given time-window that contains a history of the data streams requested by a consumer node. We 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 3 show similar trends 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, using the coded inter-variable model. 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 [13] 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.

V. Experiments

To show that our proposals achieve subtantial amounts of communication reduction, we conducted an experimental study. We analyze the effect of our coded model update (Section III) in terms of communication efficiency and we compare the performance of our coded inter-variable model (Section IV) with a state-of-the-art solution (GAMPS [13]).

A. Datasets and Cost Measures

Our experiments use two real datasets in order to contend with real phenomena: **(1) St. Bernard**: Environmental data was collected for a period of 7 months in the Grand St. Bernard area in Switzerland. We selected one deployment station and retrieved 8 distinct data streams that measured: air temperature, surface temperature, relative humidity, solar radiation, soil moisture, and wind speed. Total number of readings was 80640. Some of the streams in this dataset show quite similar trends over time. **(2) Wannengrat**: Six different physical variables were recorded over a period of two months in Wannengrat, Switzerland. The observations include relative humidity, air temperature, surface temperature, snow height, wind speed, and wind direction. Total number of readings was 104688. Unlike St. Bernard, correlations of the streams are hardly found in this dataset.

We compute the communication costs as follows: Let $|s|$ be the number of readings in an original data stream s, and $|s'|$ be the number of model updates occurred, while sweeping s. We obtain the relative communication cost for the stream s by $cost_s = \frac{|s'|\cdot|u|}{|s|\cdot 4bytes}$, where: $|u|$ is the size (in bytes) of one model update; and we used the fact that each raw reading was stored using 4 bytes.

B. Effect of Coded Model Update

In the first set of experiments, we measured the effect of using model-based views for minimizing communication cost over networks. Fig. 5 demonstrates significant improvements Our proposal achieves at best over 99 % (coded degree-0 polynomial regression, for St. Bernard) and at least 91 % (degree-2 Chebyshev regression, for St. Bernard) less data communications, compared to transmitting original data. This is expected since the coded model update compresses the model parameter values with compact bitmaps. For these experiments, the sizes of the bitmaps were only one or two bytes for each stream. To obtain these results, we set the user-given error bound to 1 %.

Next, we measured how varying error bound affects the performance of our framework. Fig. 6 shows the changes of communication cost along different sizes of error bound for degree-1 polynomial and Chebyshev regression models with

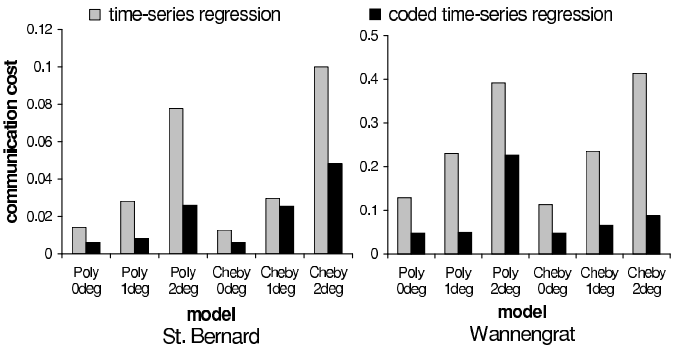

Fig. 5. Comparison of Communication Cost

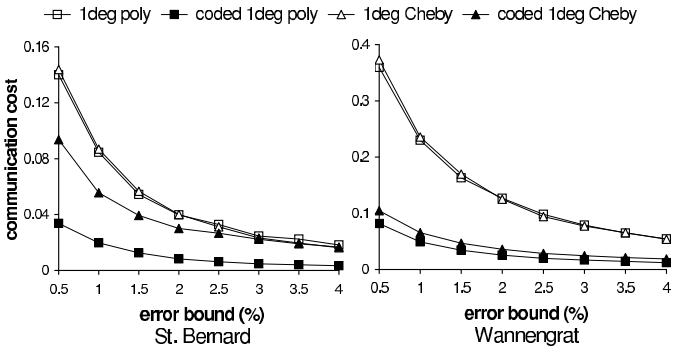

Fig. 6. Effect of Error Bound

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, as described by 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.

C. Effect of Coded Inter-Variable Model

In the second set of experiments, we measured the communication costs of our coded inter-variable model, and compared it with a state-of-the-art solution GAMPS [13]. Both are correlation models and share similar underlying ideas that exploit linear dependencies among the streams. Fig. 7 shows the results, 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 performs best, 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 IV-A, such that considering correlations of streams can reduce the redundancy of data, rendering more efficient data communication over networks.

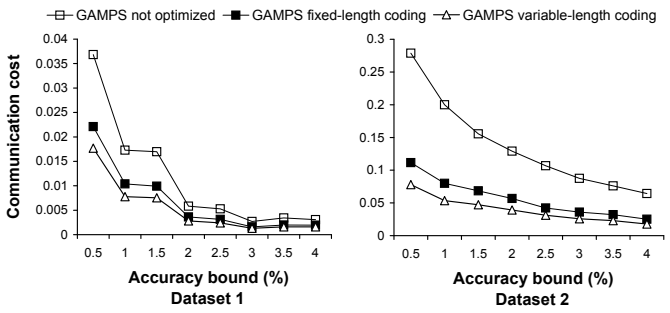

Fig. 7. Effects of Correlation Models

As we described in Section V-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. This is because the coded inter-variable model works like the piecewise constant model, when streams are uncorrelated.

VI. Related Work

Due to the characteristics of continuity, data streams are often modeled by continuous-time functions as *time-series regression models* [8], [14], [15]. 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.

Instead of building individual models for single data streams, *correlation models* for multiple streams have also been highlighted, particularly for data stream compression [12], [16], [13]. 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] 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.

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 [5], [6], [7]. 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 *).

VII. 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 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 method that boosts the performance of the framework, named *coded model update*. It compresses the parameter values of the models to be transmitted, by encoding them with compact bitmaps. We also present a *coded inter-variable model* that incorporates an effective correlation model into the efficient coded model update. Experimental results show the proposals effectively reduce communication costs.

Acknowledgements

This work was supported by the European Commission in the PlanetData NoE (contract nr. 257641), the Nano-Tera initiative (http://www.nano-tera.ch) in the OpenSense project (reference nr. 839-401), and NCCR-MICS (http://www.mics.org), a center supported by the Swiss National Science Foundation (grant nr. 5005-67322).

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. Kumar, S. Nath, *et al.*, "Environmental Monitoring 2.0," in *Proceedings of the 2009 IEEE International Conference on Data Engineering*. IEEE Computer Society, 2009, pp. 1507–1510.

[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] 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.

[5] 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.

[6] 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.

[7] 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.

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

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

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

[11] 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.

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

[13] S. Gandhi, S. Nath, S. Suri, and J. Liu, "Gamps: Compressing multi sensor data by grouping and amplitude scaling," in *Proceedings of the 35th SIGMOD international conference on Management of data*. ACM, 2009, pp. 771–784.

[14] 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.

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

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