Capturing Sensor-Generated Time Series with Quality Guarantees

Iosif Lazaridis
University of California, Irvine
Irvine, CA, USA
iosif@ics.uci.edu

Sharad Mehrotra
University of California, Irvine
Irvine, CA, USA
sharad@ics.uci.edu

Abstract

We are interested in capturing time series generated by small wireless electronic sensors. Battery-operated sensors must avoid heavy use of their wireless radio which is a key cause of energy dissipation. When many sensors transmit, the resources of the recipient of the data are taxed; hence, limiting communication will benefit the recipient as well. In our paper we show how time series generated by sensors can be captured and stored in a database system (archive). Sensors compress time series instead of sending them in raw form. We propose an optimal on-line algorithm for constructing a piecewise constant approximation (PCA) of a time series which guarantees that the compressed representation satisfies an error bound on the L_∞ distance. In addition to the capture task, we often want to estimate the values of a time series ahead of time, e.g., to answer real-time queries. To achieve this, sensors may fit predictive models on observed data, sending parameters of these models to the archive. We exploit the interplay between prediction and compression in a unified framework that avoids duplicating effort and leads to reduced communication.

1. Introduction

Data generated by small wireless electronic sensors are increasingly significant for emerging applications [14, 9, 24]. Sensors are becoming smaller, cheaper and more configurable [24]. Current and future sensor designs routinely include a fully programmable CPU, a local memory buffer and a wireless radio for communication [24, 16]. Sensors must be treated as equal partners in future distributed database systems as they can store, manipulate and communicate information.

1.1. The Time Series Capture Task

In our paper we are interested in *capturing* sensor-generated time series. Each sensor, or *data producer* generates a series of values of some measured attribute, e.g., temperature. Sending these raw values to the *data archiver* (a database system) uses up the limited communication bandwidth [23, 16] and causes energy drain [24, 20]. If multiple sources of information are involved, bandwidth at the archiver end may be limited as well [23]. Even if all information can be received, it may be too difficult to process if the rate of data generation is high [2, 30]. Obviously, limiting communication in a system involving sensors will benefit all involved parties.

We assume that some loss of precision in the archived version of the time series can be tolerated if this helps reduce communication. We do not want, however, unbounded inaccuracy in the stored imprecise series. Besides the capture task, time series values may be needed ahead of time by real-time applications, e.g., queries. Such applications and the capture task must gracefully co-exist.

We observe that time series values are not entirely random and can thus be compressed. This implies that some number of samples must be accumulated, since compression exploits the redundancy of information across many samples; the sensor must see some samples, compress them and forward the compressed representation to the archiver.

Propagating messages from the sensor to the archiver takes time. Hence, any application that requires knowledge of recent, present or future time series values must wait for these to arrive. This time will be longer if samples are not forwarded immediately but are rather compressed. To address this issue, sensors are tasked with fitting parametric predictive models of the time series, sending parameters of these models to the archive. Using these, values of the time series can be estimated ahead of time, reducing the latency seen by applications.

1.2. Why is Capturing Time Series Important?

Many applications over sensors are primarily motivated by their ability to monitor the physical world in real-time. In many situations sensor data is useful not only for its present utility for some application, but for its potential future utility as well. Therefore, capturing the complete history of a time series is essential for systems incorporating sensors. This contrasts somewhat with the emerging paradigm of rapidly produced *data streams* whose focus is not primarily on storage [20, 2].

For example, sensors will often be used in large-scale scientific experiments. Such experiments, often involving changing behavior (e.g., the diffusion of pollutants in a water stream), over long periods of time cannot be accurately studied unless one stores the entire history of the phenomenon. In some cases, e.g., major earthquakes, environmental disasters, volcano eruptions, the studied process is rare and hence the value of data collected about it is significant.

In a second example, consider an intrusion detection system relying on sound and light intensity measuring devices. A torch-carrying intruder may set off an alarm, but it is conceivable that the real-time application may be misguided into not raising an alarm. The next day, when the intrusion is detected, e.g., a precious item is missing, it would be useful to "rewind" the time series produced by the system's sensors, and try to identify traces of the intrusion.

We view time series generated by sensors as a commodity, besides its real-time usefulness. Our work is part of the Quality-Aware Sensor Architecture (QUASAR) project at UC Irvine, which aims to create a general architecture for different sensor-based applications, both on-line and off-line, both current and future. This differs from a commonplace view in which sensor-based applications are built from scratch with a single objective (e.g., real-time monitoring) without accounting for unforeseen uses of sensor-generated data.

1.3. Paper Organization

In Section 2 we formulate our problem and sketch the proposed solution. In Section 3 we consider compression with quality guarantees. In Section 4 we motivate the need for prediction, show how it can be performed, and how it can co-exist with compression. In Section 5 we evaluate our techniques experimentally. In Section 6 we cover some related work, and in Section 7 we summarize our work and present future research directions.

2. Problem Formulation

We will speak of a single data producer (sensor) and data archiver (database). Keep in mind that in a real system, the archiver will interact with many producers. Each producer-archiver interaction will use the algorithms presented in our paper. The archiver may assign different importance to different sensors. The problem of gathering data from multiple sources to achieve an overall level of quality of the archive is itself very interesting [8, 23]. Rather than capturing the time series by probing the sensor, we will do so by receiving messages from it. Wireless devices pay a heavy price (energy-wise) for listening on the radio channel even if no data is being transmitted [29].

2.1. Definitions and Assumptions

For simplicity's sake, we will assume that the producer's clock is synchronized with the archiver's. Time synchronization is an important issue of research in sensor networks [11] but goes beyond the scope of our paper. We will assume that time is discrete and will denote the *time domain* as $\mathcal{T} = \{1,\ 2,\ \ldots\}$. The time quantum, corresponding to one step, is the sampling period of the sensor. We will also deal with time series whose *value domain* is $\mathbb{R}$, i.e., the real numbers.

We define a time series as a sequence $S = \langle s[1], s[2], \ldots \rangle$ where $s[k] \in \mathbb{R}$ is the *value* of an observed real-world process at *time position* $k \in \mathcal{T}$. We note the time position of *now* as n. The *observed series*, at time n is noted as $S^n = \langle s[1], s[2], \ldots, s[n] \rangle$. We use $s[i : j]$ to note a subseries from time i to time j, i.e., $s[i : j] = \langle\ s[i],\ s[i+1],\ \ldots\ ,\ s[j-1],\ s[j]\ \rangle$. Hence, $S^n = s[1 : n]$.

The sensor has a finite energy supply. This is depleted during normal sensing operation at some rate. Additional energy drain is caused when using any of the sensor's equipment, including (i) powering its memory, (ii) using its CPU, (iii) communicating bits, or listening on the wireless channel. The specific rates of energy consumption for these operations are sensor-specific. Modern sensors try to be power-aware, shutting down components when they are not used or e.g., changing the frequency/voltage of the processor [15] depending on the workload.

Different sensors are bound to differ in the details of their architecture and power consumption. We simply observe that communication is often the major cause of energy drain [24, 16] in sensors and hence, in the interest of extending the sensor's life, communication must be curtailed.

2.2. Objectives

We identified communication as the main target for optimization. Our goals are the following:

- To *capture* an approximate version $\widehat{S}$ of the series S in the archive. $\widehat{S}$ is conceptually a time series for the same time positions as S but has a smaller actual representation. We present algorithms to construct such a *compressed* representation in Section 3.

- To *predict* values of S ahead of time, i.e., before receiving them at the archive. This can be achieved by fitting predictive models, using these to estimate values. We examine the problem of prediction in our setting in Section 4.

2.3. Quality Metric

A commonly used metric for comparing time series is that of *Euclidean distance* [17, 12]. If $S^n = s[1:n]$ and $\widehat{S}^n = \widehat{s}[1:n]$ then this is defined as:

$$d(S^n, \widehat{S}^n) = \frac{1}{n} \sum_{k=1}^{n} (s[i] - \widehat{s}[i])^2$$

If we were to specify quality as an upper bound on this distance, then we would make room for large divergence of individual samples of the time series. For similarity-retrieval types of applications [26, 17], the main goal is to identify similarity of overall structure, rather than similarity of individual values. We do not want to assume the use of the time series data. Hence, we will use a *stronger* notion of quality, namely that the estimation for any individual sample $\widehat{s}[k]$ should not deviate from $s[k]$ more than an upper bound. We can use the L_∞ metric:

$$L_\infty(S^n, \widehat{S}^n) = \max_{1 \le k \le n} |s[k] - \widehat{s}[k]|$$

and state the quality requirement as follows:

$$L_\infty(S^n, \widehat{S}^n) \le \epsilon_{capt} \Leftrightarrow \max_{1 \le k \le n} |s[k] - \widehat{s}[k]| \le \epsilon_{capt}$$

$\widehat{S}^n$ is said to be a within-ϵ_{capt} approximation of S^n if the above holds. Note that this is a "stronger" notion of quality in that it implies a bound on the Euclidean distance[1]:

$$L_\infty(S^n, \widehat{S}^n) \le \epsilon_{capt} \Rightarrow d(S^n, \widehat{S}^n) \le \epsilon_{capt}^2$$

Our first objective can be formalized as capturing and storing a within-ϵ_{capt} version of S at the archive, where ϵ_{capt}, is a user-specified *capture*, or *compression tolerance*.

2.4. Latency and the need for prediction

The use of prediction is motivated by the latency between the producer and the archiver of the time series. This can be broken down to:

- *Communication Latency, n_{comm}* .— This includes the transmission, propagation and queuing times in both the wireless and wired links between producer and archiver.

[1]Actually $d(S^n, \widehat{S}^n) \le \epsilon$ also implies a bound on $L_\infty(S^n, \widehat{S}^n)$, namely that $L_\infty(S^n, \widehat{S}^n) \le \sqrt{n\epsilon}$, but this is a very loose bound as it is proportional to $\sqrt{n}$.

- *Compression Latency, n_{comp}* .— This consists of the time spent at the sensor processing S^n so as to produce $\widehat{S}^n$.

As a result of the overall latency, at time n, the archiver will have received not $\widehat{S}^n$, but rather $\widehat{S}^{n-n_{lag}}$ where $n_{lag} = n_{comm} + n_{comp}$ is the number of time positions it is "behind" the producer. Any real-time applications (e.g., queries) that require the value of the time series for any time position n_q, in the future ($n_q > n$), the present ($n_q = n$), or the recent past ($n - n_{lag} < n_q < n$) must wait for that value to arrive from the producer.

Suppose that the value of the time series at time position $n_q > n - n_{lag}$ is needed. The system will provide an estimate $s_q[n_q]$ using any of the following three evaluation strategies:

- *Predict*.— Some predictive model M and its parameters $\underline{\theta}$ are stored in the archive. Subsequently, $s_q[n_q]$ is predicted as $\widetilde{s}_{M,\underline{\theta}}[n_q]$. We will note this as $\widetilde{s}[n_q]$ when $(M, \underline{\theta})$ is known. This raises the issue: how does one obtain such a predictive model, and how can one be guaranteed of the difference between the predicted and the actual value.

- *Probe*.— The producer is asked directly for $s[n_q]$. This requires from the producer to maintain all samples (or at least their approximations) it has not forwarded to the archive. Additionally, the producer must now listen on the wireless channel for probes. This is a cause of energy drain [29]. The producer can tune in to listen for probes occasionally. This will, however, increase the latency seen by queries.

- *Wait*.— The final strategy is to simply wait for $\widehat{s}[n_q]$ to arrive from the producer. The quality of $s_q[n_q] = \widehat{s}[n_q]$ is guaranteed; it is within ϵ_{capt} of $s[n_q]$.

We note here that prediction is very attractive, since it does away with the latency involved with either doing a probe or waiting for a value to be sent by the sensor. Our second objective can be formalized as providing, to any interested applications, a within-ϵ_{pred} estimation of time series values, *before* these values arrive at the archive. In Section 4 we show how this can be achieved. We will also briefly discuss the important problem of choosing a predictive model among many, and present the criterion by which different candidate models can be compared.

2.5. Combining Compression with Prediction

Some of the work done for prediction can be used for the capture task as well. If the predictive model is somewhat accurate, then the archive already has an idea of some time series values before receiving them. The sensor can use this to limit the effort that must be spent to compress the time series. In Section 4.4 we will show how this basic intuition can be used algorithmically.

3. Compression Algorithms

Work in approximating time series has been extensive in the literature. Time series have been approximated using wavelets [4], Fourier transforms [1], piecewise linear approximations [18], or polynomials [26]. Since the approximation must be carried out by the sensor, a device of limited abilities, the employed algorithm must be lightweight in terms of processing and memory utilization.

3.1. The Piecewise Constant Approximation

An attractive type of lossy[2] compression is the *piecewise constant approximation*[3] (PCA) [17], whereby the time series S is represented as a sequence of K segments:

$$PCA(S^n) = \langle\, (c_1,\, e_1),\, (c_2,\, e_2),\, \ldots\, ,\, (c_K,\, e_K)\,\rangle$$

where e_k is the end-point of a segment and c_k is a constant value for times in $[e_{k-1}+1,\, e_k]$, or for times in $[1,\, e_1]$ for the first segment. In such an approximate representation, we estimate $s[k]$ as:

$$\hat{s}[k] = \begin{cases} c_1 & \text{if } k \le e_1 \\ c_m & \text{if } e_{m-1}+1 \le k \le e_m \end{cases}$$

This representation is intuitive and easy to index. In terms of compression, we note that a single segment costs us $b_s + b_{tp}$ to store, where b_s is the size of a sample value and b_{tp} is the size of a time position index. If a time series of length n is approximated with a PCA sequence of length K, then the compression ratio is $\frac{K(b_s+b_{tp})}{nb_s}$. If each segment corresponds to many samples of the time series ($\frac{K}{n}$ is significantly less than 1), then high compression ratios can be achieved.

A series S^n can be approximated by different $PCA(S^n)$ approximations. As we will see, very simple on-line algorithms with $O(1)$ space requirement can be used to construct a PCA representation that preserves the desired quality guarantee with minimum K.

3.2. Poor Man's Compression

Poor Man's Compression (PMC) is a bare-bones form of compression that can be used to reduce the size of a time-series representation. It is an on-line algorithm, producing

[2]Experimentation with lossless methods (gzip, not reported) indicate very small ($\sim$50%) compression ratios. Lossless compression also does not exploit the precision-compression tradeoff.

[3]This was called *Adaptive Piecewise Constant Approximation* (APCA) in [17] to distinguish it from a similar approximation (PAA) with equal segment lengths.

```
procedure PMC-MR
INPUT:
time series $S = \langle s[1],\ s[2],\ \ldots\ \rangle$, tolerance $\epsilon_{capt} > 0$.
OUTPUT:
compressed time series PCA(S) within-$\epsilon_{capt}$ of $S$.

(1)  PCA(S) ← ⟨ ⟩;
(2)  n ← 1;
(3)  m ← s[n];
(4)  M ← s[n];
(5)  while $S$.hasMoreSamples()
(6)    if $\max\{M,\ s[n]\} - \min\{m,\ s[n]\} > 2\epsilon_{capt}$
(7)      append $(\frac{M+m}{2},\ n-1)$ to PCA(S);
(8)      m ← s[n];
(9)      M ← s[n];
(10)   else
(11)     m ← $\min\{m,\ s[n]\}$;
(12)     M ← $\max\{M,\ s[n]\}$;
(13)   end;
(11)   n ← n+1;
(12) end;
(13) append $(\frac{M+m}{2},\ n-1)$ to PCA(S);
```

Figure 1. PMC-MR Algorithm

segments of the PCA representation as new samples arrive. It requires only $O(1)$ space and performs $O(1)$ computation per sample. Hence, its overall time complexity for a series of size n is $O(n)$. This computation is interspersed with the arrival of samples; the compressed series is "ready to go" as soon as the last sample is processed. Hence the n_{comm} time of Section 2 is minimized.

Let $s[i:j]$ be some time series. Can this be compressed in a single segment in a manner that preserves the ϵ_{capt} guarantee? Lemma 1 supplies the necessary and sufficient condition.

Lemma 1 *The time series $s[i:j]$ can be compressed in a single segment $(c,\ j)$ with an error tolerance ϵ_{capt} iff:*

$$range[i:j] = \max_{i\le k\le j} s[k] - \min_{i\le k\le j} s[k] \le 2\epsilon_{capt}$$

Proof: If for all $k : i \le k \le j : |c - s[k]| \le \epsilon_{capt}$ then also $|c - \max_{i\le k\le j} s[k]| \le \epsilon_{capt}$ and $|c - \min_{i\le k\le j} s[k]| \le \epsilon_{capt}$. Hence, $|\max_{i\le k\le j} s[k] - \min_{i\le k\le j} s[k]| \le 2\epsilon_{capt}$, or $range[i:j] \le 2\epsilon_{capt}$. We used the proposition $|a| \le b \wedge |c| \le d \Rightarrow |a-c| \le b+d$. Conversely, if $range[i:j] \le 2\epsilon_{capt}$ then the segment $(\frac{\max_{i\le k\le j} s[k] + \min_{i\le k\le j} s[k]}{2},\ j)$ can be trivially shown to compress it within-ϵ_{capt}.

3.2.1. Poor Man's Compression - Midrange. Our first algorithm (see Figure 1), PMC-MR (for midrange) uses the converse of Lemma 1. It monitors the range of its input. While this is less or equal to $2\epsilon_{capt}$ it updates the range if needed (lines 11-12). When the range exceeds $2\epsilon_{capt}$ at time n, then the segment ending at $n-1$ with a constant being equal to the midrange of the preceding points is output (line 7). The algorithm then tries to compress the next set of samples, starting at time n (lines 8-9).

PMC-MR not only achieves the goal of compression, but satisfies an even stronger property: that *no* other PCA representation satisfying the ϵ_{capt} constraint, over *any* input time series can be a valid compression for that time series if it has fewer segments. PMC-MR is thus *instance optimal* not only for the class of on-line algorithms, but over *any* algorithm that solves this problem correctly. We state our claim and its proof formally.

Theorem 1 *Let $S^n = s[1 : n]$ be an aribitrary time series that must be approximated with a piecewise constant approximation that satisfies for all $k = 1, 2, \ldots, n$ that $|\hat{s}[k] - s[k]| \leq \epsilon_{capt}$. If the PMC-MR algorithm (Figure 1) produces a $PCA(S^n)$ representation with K segments, then no valid PCA representation with $K' < K$ segments exists.*

Proof: By contradiction. Let $BETTER(S^n)$ be a valid representation of S^n with $K' < K$ segments. Hence, $K = K' + m$ where $m \geq 1$. Therefore of the K intervals of $PCA(S^n)$ at least one does not contain an endpoint of $BETTER(S^n)$. This cannot be the final one, since that must be n, which is contained in the final interval of $PCA(S^n)$. Let $[e_{i-1}+1,\ e_i]$ be the interval of $PCA(S^n)$ that does not contain an endpoint of $BETTER(S^n)$. Let $[e_{i-1}+1-v,\ e_i+w]$ with $v \geq 1, w \geq 1$ be the interval of $BETTER(S^n)$ that covers $[e_{i-1}+1,\ e_i]$ Thus, $[e_{i-1}+1-v,\ e_i+w]$ covers $[e_{i-1}+1,\ e_i+1]$ as well (since $w \geq 1$). But since $[e_{i-1}+1,\ e_i]$ is an interval of $PCA(S^n)$, produced by the PMC-MR algorithm, and it is not the final one, then the range of values in $[e_{i-1}+1,\ e_i+1]$ is greater than $2\epsilon_{capt}$. The range of values in any time interval is always greater than the range of values in any of its subintervals. Hence, the range of values in $[e_{i-1}+1-v,\ e_i+w]$ is greater than $2\epsilon_{capt}$. Therefore there doesn't exist a value c such that for all values $s[k]$ where $k \in [e_{i-1}+1-v,\ e_i+w]$ it is $|c - s[k]| \leq \epsilon_{capt}$ (Lemma 1). Therefore, the segment of $BETTER(S^n)$ whose endpoint is at time position $e_i + w$ violates the ϵ_{capt} constraint and $BETTER(S^n)$ is not a valid representation for S^n. ▮

PMC-MR does an optimal job at compression, but it has two disadvantages. First, the time series it generates cannot be easily incorporated in similarity-retrieval systems, which usually rely on PCA representations where the constant value for each segment is the mean of the time series values for that segment [17]. Second, the mean error produced by PMC-MR may sometimes be large, even close to ϵ_{capt}, especially if the distribution of values is skewed. This problem does not conflict with our specification of quality, but it is an undesirable property of the algorithm.

Consider the time series $S = \langle\ 0,\ 0,\ 0,\ 4\ \rangle$ and suppose that $\epsilon_{capt} = 2$. PMC-MR will approximate it with one segment $(2,\ 4)$. The mean error will be $\frac{|0-2|+|0-2|+|0-2|+|4-2|}{4} = 2$.

3.2.2. Poor Man's Compression - Mean. To address these problems, we propose a modified algorithm, called Poor Man's Compression-Mean (PMC-MEAN). PMC-MEAN is identical to PMC-MR except that it uses the mean of the points in each segment as the constant of the segment. Values are sampled until the mean of the points seen thus far is more than ϵ_{capt} away from the minimum or maximum of the observed points. Then, a segment is output and the algorithm tries to compress the next set of points starting from the one that caused the tolerance violation.

As an example, for the series S above, PMC-MEAN would output two segments $(0,\ 3)$ and $(4,\ 4)$. Its error would be zero for these segments, but it will have produced more segments than the optimal algorithm (PMC-MR). Choosing between these two algorithms must depend on the use of the data and their relative performance at compression. In our experiments of Section 5 we will see that over many datasets, PMC-MEAN performed only little worse than PMC-MR. Hence, we consider it as a viable alternative to PMC-MR.

3.2.3. PCA Segment Transmission. We observe that the PMC algorithms produce a sequence of compressed segments. These can be forwarded either immediately, or aggregated into packets for network transmission when either the sensor's memory buffer is filled, or the maximum packet length is reached. Normally, we would like to fit as many segments into a packet as possible, since each packet has some overhead in terms of header information. However, since packets may be lost, especially over the unreliable links available to sensors, it might make sense to limit the maximum packet length for transmission, thus avoiding the loss of large segments of the time series all at once.

Note, that there is no guarantee for the time it takes for a segment to be output. If compression is successful, then potentially, a single segment could go on forever — if all new points do not cause the violation of the ϵ_{capt} condition. In practice, we might interrupt these algorithms if we want to receive segments of the time series in a timely manner.

4. Prediction

In Section 2, we motivated the use of prediction from the need of real-time applications to co-exist with the capture task. We will now address some issues arising when prediction is performed.

4.1. Who should predict?

There are two fundamental ways in which prediction can be used:

- *Archive-side.*— The data archive contains at least $\widehat{s}[1:n-n_{lag}]$. This can be used, via some prediction model, to provide an estimate $\widetilde{s}[k]$, for time positions $k > n - n_{lag}$.
- *Producer-side.*— The producer sees the entire $s[1:n]$. Hence, it can also use some prediction model to provide an estimate $\widetilde{s}[k]$. In this case, the parameters of this model need to be transmitted to the archive.

Archive-side prediction has the obvious advantage of not requiring communication with the sensor[4]. A second advantage is that the archive sees the "broad picture" of the sensor's history. It can thus infer predictive models at a larger time scale, accounting perhaps for cyclic behavior, global trends or other aspects not discernible from the sensor's limited (time-wise) perspective. Its disadvantages are: (i) it is based on $\widehat{S}$ and not on the precise S, (ii) it cannot provide any prediction quality guarantee, as the archive does not monitor the precise S which can deviate from the predicted $\widetilde{S}$ without bound, and (iii) prediction must be accurate n_{lag} steps into the future for it to predict the present value accurately. As we mentioned in Section 3.2.3, n_{lag} may be very large.

Producer-side prediction has the disadvantage of requiring communication. Since producers have limited memory, only the most recent past of the time series can be stored in it, or perhaps very coarse derivative information about the more distant past. Hence, long-term effects like cycles cannot be incorporated into the prediction model. However, the main advantage of producer-side prediction is that it uses the raw S series, and allows for prediction guarantees. Producer-side prediction will be used in the following.

4.2. Producer-Side Prediction

The basic form of Producer-Side Prediction (PSP) is shown in Figure 2. The input of the algorithm is a time series, a prediction tolerance ϵ_{pred} and a parametric predictive model M. PSP begins by guessing a set of parameters for M (line 1). Subsequently, each sample $s[k]$ is checked against the predicted value based on the last set of parameters $\underline{\theta}^{last}$ (line 7). If this is greater than ϵ_{pred}, then a new set of parameters is computed (line 8), by updating the old parameters with the samples at time positions greater than the time when the last prediction parameters were estimated. The algorithm produces a sequence of $(\underline{\theta}, n)$ pairs. These predict the time series starting from time n. This sequence defines a within-ϵ_{pred} approximate version of S, which we may note as $\widetilde{S}$.

```
procedure PSP
INPUT:
time series $S = \langle\ s[1],\ s[2],\ \ldots\ \rangle$,
prediction tolerance $\epsilon_{pred} > 0$, model $M$.
OUTPUT:
prediction sequence $PS = \langle(\underline{\theta}_1,\ t_1),\ (\underline{\theta}_2,\ t_2),\ \ldots\rangle$.
(1)  guess $\underline{\theta}$ for M;
(2)  PS $\leftarrow \langle\ (\theta,\ 0)\ \rangle$;
(3)  $\underline{\theta}^{last} \leftarrow \underline{\theta}$;
(4)  $n^{last} \leftarrow 0$;
(5)  n $\leftarrow$ 1;
(6)  while S.hasMoreSamples()
(7)    if $|\widetilde{s}_{M,\underline{\theta}^{last}}[n] - s[n]| > \epsilon_{pred}$
(8)      $\underline{\theta} \leftarrow$ updateParameters($M,\ \underline{\theta}^{last},\ s[n^{last}+1:n]$)
(9)      append $(\underline{\theta},\ n)$ to PS;
(10)     $\underline{\theta}^{last} \leftarrow \underline{\theta}$;
(11)     $n^{last} \leftarrow n$;
(12)   end;
(13)   $n \leftarrow n + 1$;
(14) end;
```

Figure 2. Producer-Side Prediction with Error Guarantee

We observe that prediction parameters do not arrive instantaneously to the archive. Let τ be an upper bound on this time. Clearly, if the time were unbounded, then PSP would provide no guarantee, as queries can never be certain whether a parameter refresh is on its way or not.

4.2.1. Setting ϵ_{pred}. The best value for ϵ_{pred} depends on the quality requirements of real-time applications. If values must be predicted frequently at a high quality, then ϵ_{pred} must be set low. This problem was studied in detail in [22]. In that paper, the (implicit) prediction model was $\widehat{s}[k] = s[n^{last}]$. We can adopt a similar algorithm to set ϵ_{pred} adaptively. The main intuition in [22] is that as data becomes more variable, ϵ_{pred} is increased, to reduce the number of messages. On the other hand, when queries arrive at a high rate with small error tolerances, ϵ_{pred} is decreased to make sure that these queries can be answered at the server without performing any probes. Setting ϵ_{pred} adaptively does not conflict with the algorithms presented in this paper.

4.2.2. Choosing a Prediction Model. Our use of prediction does not assume a particular predictive model. The actual model to be used, must be chosen based on (i) domain knowledge about the monitored attribute, and (ii) engineering concerns, i.e., the cost associated with fitting prediction models and (especially) transmitting their parameters. Traditionally, prediction performance is

[4]In principle, the archive could also send prediction parameters to the sensor, especially if prediction guarantees are required.

gauged by prediction error. Suppose that $(M_1,\ \underline{\theta}_1)$ and $(M_2,\ \underline{\theta}_2)$ are competing models with their parameters. If at time n, it is the case that:

$$|s[n] - \widetilde{s}_{M_1,\underline{\theta}_1}[n]| > |s[n] - \widetilde{s}_{M_2,\underline{\theta}_2}[n]|$$

then $(M_1,\ \underline{\theta}_1)$ is a "worse" predictor than $(M_2,\ \underline{\theta}_2)$ at time n.

From a system performance perspective, as long as $(M_1, \underline{\theta}_1)$ and $(M_2, \underline{\theta}_2)$ do not produce errors greater than ϵ_{pred}, (resulting in new transmission of new parameters), they are equivalent. Consider competing models $M_1,\ M_2$ and let $|\underline{\theta}_1|,\ |\underline{\theta}_2|$ denote the size (in bytes) of their parameters. If K_1 messages are generated by M_1 and K_2 by M_2, then M_1 is preferred if $K_1|\underline{\theta}_1| < K_2|\underline{\theta}_2|$, since this leads to reduced data transmission.[5] If the model must be fixed a priori, then a decision must be made based on the above criterion, using experimentation, expert opinion or past experience to choose between competing models.

4.2.3. Adaptive Model Selection. In many situations, a global model for predicting a time series is not a valid assumption. It is likely that a time series can best be approximated by different models for different *epochs* of its history. We informally define an *epoch* as a time period during which a time series' behavior is well-predicted by a single model.

Consider for example a moving object in one dimension. The object's position at different times is the time series of interest. At times, the object may be idle. The best model is then $\widetilde{s}[n] = c$. At other times, it is moving at a constant speed; a good model is then $\widetilde{s}[n] = v \cdot (n - n^{last}) + s[n^{last}]$. Sometimes it is accelerating, or decelerating, etc. All these times are epochs of the object's history.

The problem of detecting changes in sequences is complex [27, 13]. A general solution, applicable to different types of time series and different classes of models cannot be easily found. The two main problems in epoch detection is to (i) discover the epoch boundaries, and (ii) not be fooled by temporary abnormal behavior into thinking that a new epoch has commenced. These matters are part of our current research. Briefly, we anticipate two modes of adaptivity:

- *Producer-side model selection*, in which the sensor chooses from competing models in reaction to the changing behavior of the time series.
- *Archiver-side model selection*, in which the archive monitors system performance and "uploads" prediction code to the sensor, either by expert intervention, or automatically.

[5]Depending on the network protocols in place, the difference of $|\underline{\theta}_1|$ vs. $|\underline{\theta}_2|$ may not be critical. Costs associated with the protocols (e.g., headers) may dominate the cost of transmitting either $\underline{\theta}_1$ or $\underline{\theta}_2$, if e.g., the difference between $|\underline{\theta}_1|$ and $|\underline{\theta}_2|$ is only a few bytes. In such a case, the simple test $K_1 < K_2$ would be preferred.

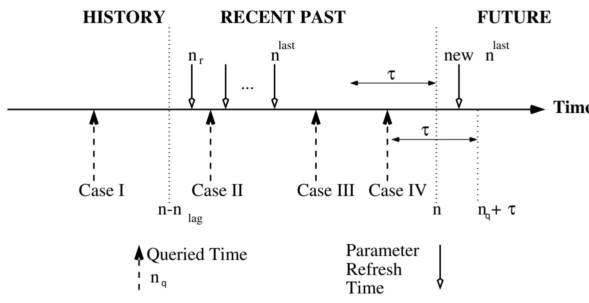

Figure 3. Estimating Time Series Values

In Section 5 we will perform a simple experiment validating the need for adaptive model selection.

4.3. Estimating time series values ahead of time

An application q arrives at time n, asking for some estimate $s_q[n_q]$ of $s[n_q]$ with bounded error: $|s[n_q] - s_q[n_q]| \leq \epsilon_q$. The discussion that follows will refer to Figure 3.

- *Case I*: $n_q \leq n - n_{lag}$. The estimate is based on the captured series: $s_q[n_q] = \widehat{s}[n_q]$. If $\epsilon_q < \epsilon_{capt}$, then the quality of the captured series does not suffice to provide an estimate with error bounded by ϵ_q. In such a case, the system can only provide an estimate $s_q[n_q] = \widehat{s}[n_q]$ which may violate the ϵ_q tolerance.

  Let $(\underline{\theta}^{last},\ n^{last})$ be the most recent parameters received at the archive.

- *Case II*: $n^{last} \geq n^q$. We look at the prediction sequence for the parameters $(\underline{\theta},\ n_r)$ where n_r is the most recent time position (with respect to n_q) in which parameters were refreshed. The answer is $s_q[n_q] = \widetilde{s}_{M,\underline{\theta}}[n_q]$ if $\epsilon_{pred} \leq \epsilon_q$. If $\epsilon_q < \epsilon_{pred}$, but $\epsilon_q \geq \epsilon_{capt}$, then the application can wait for $\widehat{s}[n_q]$ to arrive. As we have mentioned in Section 3.2.3, this time may be unbounded. We can force compressed segments to be sent in a timely manner. Finally, if $\epsilon_q < \epsilon_{capt}$, then it is necessary to do a probe which returns the exact[6] $s[n_q]$.

- *Case III*: $n^{last} < n_q$ and $n - n_q > \tau$. The answer is $s_q[n_q] = \widetilde{s}_{M,\underline{\theta}^{last}}[n_q]$. If at time n^q, the time series had violated the ϵ_{pred} tolerance, then we would have received new parameters by "now" (n). As we have not received new parameters (since $n^{last} < n_q < n$)

[6]Samples need to be kept in the sensor's buffer if applications are allowed to probe the sensor for exact values.

then, we are guaranteed that $|s[n_q] - s_q[n_q]| \leq \epsilon_{pred}$. Again, if $\epsilon_q \leq \epsilon_{pred}$, a probe has to be issued for the exact $s[n_q]$.

- *Case IV*: $n^{last} < n_q$ and $n - n_q < \tau$. Potentially, new parameters have been estimated for time n_q. The last update is not guaranteed to be valid until time position $n_q + \tau$. Thus, we wait until that time. While we wait, it may be that n^{last} changes, since parameters estimated at times both before n_q and after n_q may arrive. As long as they are before n_q, we still have to wait. Otherwise (new n^{last} of Figure 3), it becomes that $n^{last} > n_q$, in which case we estimate the value as described (Case II). Once again, we can choose to wait for $\widetilde{s}[n_q]$ or probe for $s[n_q]$ depending on the ϵ_q.

4.4. Combining Prediction and Compression

In our discussion so far, we have treated compression and prediction separately. However, both of them estimate values of the same time series, from its past and its future respectively. Can we further reduce the communication cost by combining the two?

Observe, that prediction in itself can be viewed as a form of compression. Each set of parameters is a within-ϵ_{pred} approximation of the time series for all times until the next parameter refresh. If we wanted to capture the time series within ϵ_{capt}, we could just as well have set $\epsilon_{pred} = \epsilon_{capt}$. This would have sufficed to capture the time series in the archive.

How good would is such a strategy? If the time series is very predictable, it may work better than any form of compression that doesn't assume a model for the time series. Consider, e.g., the time series $s[n] = n$. By fitting the model, $\widetilde{s}[n] = n$, we never need to re-send any parameters at all. On the other hand, suppose that the model approximates the time series behavior poorly. Then, parameter refreshes would be sent frequently. Compression would work much better in this case.

Clearly, if $\epsilon_{pred} \leq \epsilon_{capt}$, no compression is needed. If $\epsilon_{pred} > \epsilon_{capt}$ then $\widetilde{s}[n]$ may deviate from $s[n]$ by more than ϵ_{capt}. We can still make use of $\widetilde{s}[n]$ by observing the following result.

Theorem 2 *Let* $\Delta^n = \delta[1:n] = \langle\, s[1] - \widetilde{s}[1],\ s[2] - \widetilde{s}[1],\ \ldots,\ s[n] - \widetilde{s}[n]\,\rangle$. *If* $\widehat{\Delta}^n = \widehat{\delta}[1:n]$ *is a within-*ϵ_{capt} *approximation of* Δ^n, *then the series* $\widehat{S}^n = \langle\, \widetilde{s}[1] + \widehat{\delta}[1],\ \widetilde{s}[2] + \widehat{\delta}[2],\ \ldots,\ \widetilde{s}[n] + \widehat{\delta}[n]\,\rangle$ *is a within-*ϵ_{capt} *approximation of* S^n.

Proof: By contradiction. Let $\widehat{S}^n$ not be a within-ϵ_{capt} approximation of S^n. Then, there exists some k such that $|s[k] - \widehat{s}[k]| > \epsilon_{capt} \Leftrightarrow |s[k] - \widetilde{s}[k] - \widehat{\delta}[k]| > \epsilon_{capt} \Leftrightarrow |\delta[k] - \widehat{\delta}[k]| > \epsilon_{capt}$. This contradicts our hypothesis, because $\widehat{\Delta}^n$ is a within-ϵ_{capt} approximation of Δ^n. ∎

Theorem 2 presents an alternative strategy for compressing S, if prediction is performed in the system. The sensor can monitor the time series Δ of the prediction errors and compress it within ϵ_{capt}. Subsequently, the compressed $\widehat{\Delta}$ can be sent to the archive which can then obtain a within-ϵ_{capt} version of S by adding the predicted series $\widetilde{S}$ to the compressed error $\widehat{\Delta}$.

When is compressing Δ preferrable to compressing S? This depends on ϵ_{pred} and the quality of the predictive model. When ϵ_{pred} is close to ϵ_{capt} then the error series Δ, taking values in the interval $[-\epsilon_{pred}, \epsilon_{pred}]$ will probably violate the ϵ_{capt} tolerance infrequently. Hence, compressing Δ may be better than compressing S. Conversely, as ϵ_{pred} increases, the error is allowed to fluctuate more; so, perhaps compressing S is preferrable. The quality of the predictive model is also a major factor. If Δ has a small range (irrespective of ϵ_{pred}) then it may be more compressible than S.

In Figure 4 we show (first four plots), a time series S, its within-$\epsilon_{capt} = 2$ compression $\widehat{S}$, its within-$\epsilon_{pred} = 2.5$ prediction $\widetilde{S}_{2.5}$ and its within-$\epsilon_{pred} = 5$ prediction $\widetilde{S}_5$. Subsequently, we show (next four plots)the prediction error $\Delta_{2.5}$, its compression $\widehat{\Delta}_{2.5}$, the prediction error Δ_5, its compression $\widehat{\Delta}_5$. The compression of S has 14 segments, while the compression of $\widehat{\Delta}_{2.5}$ has only 10 segments, since the prediction tolerance $\epsilon_{pred} = 2.5$ is close to $\epsilon_{capt} = 2$. For this time series, the compression of $\widehat{\Delta}_5$ has 16 segments and is thus worse than the compression of S.

In Section 5 we will see situations where either compressing S, or Δ is better. The sensor can compress the series in both ways. When a message is to be transmitted, it can choose to forward either the compressed series or the compressed error, depending on which is smaller. This has a small overhead for adding a marker to identify the used strategy and a 2-fold increase in processing and memory usage at the sensor. This is reasonable, since it reduces communication.

5. Performance Study

In this section, we perform an evaluation of our ideas in this paper. The results confirm the good performance of our algorithms under different situations.

5.1. Compression Experiments

First, we examine the effectiveness of PMC-MR and PMC-MEAN for synthetic and real-world data. We use synthetic *Random Walk* data generated as:

$x[1] = 0$ and $x[n] = x[n-1] + s_n$ where $s_n \sim U(-1, 1)$

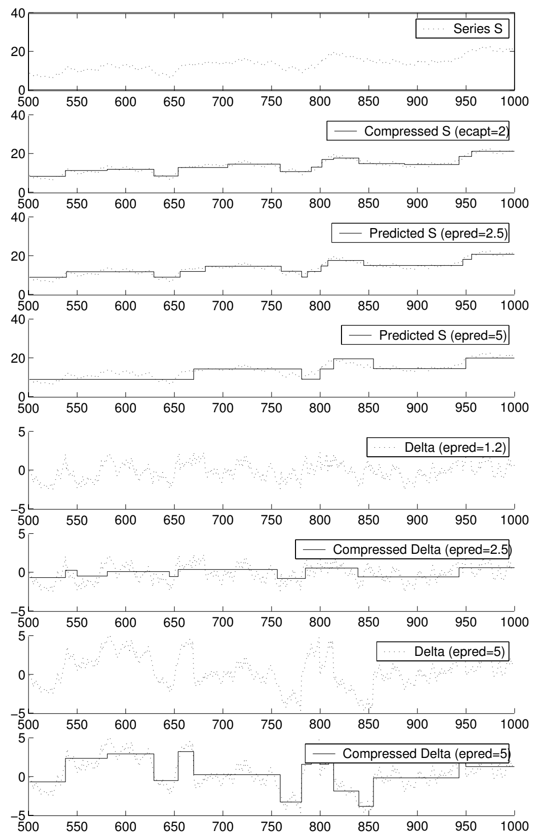

Figure 4. Combinng Prediction and Compression

Dataset	n	μ	σ	$range[1:n]$
Random Walk	100,000	42.50	59.80	[-53.55, 148.35]
Sea Surface Temperature	143,508	28.62	0.67	[25.82, 31.87]
Salinity	54,531	34.75	0.26	[33.41, 35.28]
Shortwave Radiation	117,069	269.41	358.00	[0, 1351.3]

Table 1. Statistics for Time Series used in our Compression Experiments

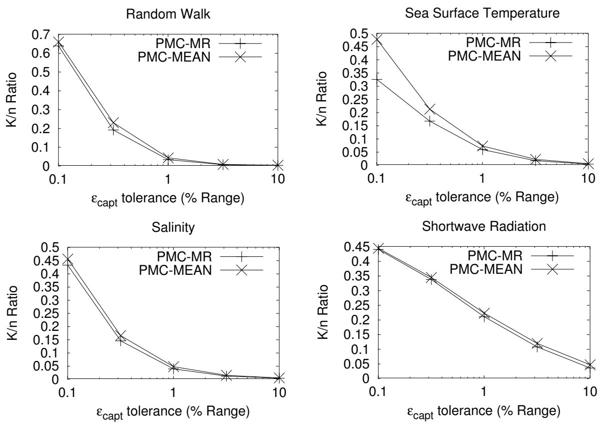

Figure 5. Compression Performance (K/n ratio)

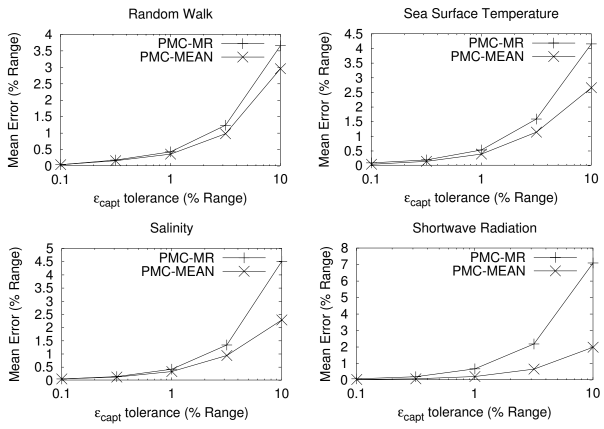

Figure 6. Compression Performance (Mean Absolute Error)

We also used time series of environmental variables from an oceanographic buoy sampled at 10 min intervals [21]. We used a *Sea Surface Temperature*, *Salinity*, and *Shortwave Radiation* series. Statistics about all used series are given in Table 1. We preprocessed the buoy series to remove missing values. We compress these time series at various ϵ_{capt}. We chose ϵ_{capt} as follows. We first determined the range of each time series and used 1/1000th of that as our baseline ϵ_{base}. We compressed the time series ϵ_{capt} by multiplying ϵ_{base} with factors of $\sqrt{10}$. We thus covered compression tolerances from $\frac{1}{1000}$ to $\frac{1}{10}$ of the time series value range.

In Figure 5 we show the $\frac{K}{n}$ ratio achieved by PMC-MR and PMC-MEAN over these time series for varying ϵ_{capt}. As expected, this ratio drops as ϵ_{capt} increases. The performance of PMC-MEAN is very slightly worse than the optimal PMC-MR algorithm. For the central ϵ_{capt} value (1% of range), the $\frac{K}{n}$ ratio was on average 8.3% for PMC-MR and 9.4% for PMC-MEAN.

In Figure 6 we show the mean absolute error over all time positions. This is roughly less than half the ϵ_{capt} maximum. PMC-MEAN and PMC-MR were comparable over the unbiased synthetic data, but with real data, PMC-MEAN had a slight edge. This is due to better approximation by using the mean, and to the greater number of segments output by PMC-MEAN for the same ϵ_{capt}.

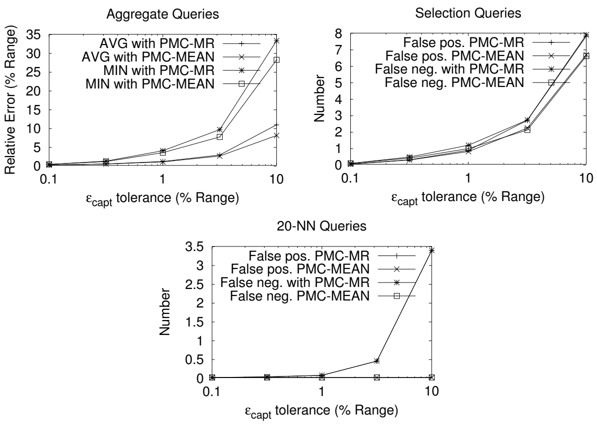

Figure 7. Answering Queries over Compressed Time Series

Next, we test how query performance is impacted by using compressed as opposed to precise time series. We generate 100 series, each with 1,000 time positions from the random walk model, choosing the $x[1] \sim U(0, 10)$ to simulate the difference in values reported by different sensors. We compressed these using the $\epsilon_{base} = 0.2019$ and its $\sqrt{10}$ multiples as before, and asked: (i) 100 queries, for random time positions of the form "What is the minimum and average sensor reading?" (Aggregate Queries), (ii) 1,000 queries, one per time position of the form "Which sensors' values are above c?" (Selection Queries), where c is uniformly chosen from $[0, 10]$ for each query, and (iii) 100 queries, asking for the 20 nearest neighbors, in terms of Euclidean distance, of all 100 time series (20-NN Queries).

The results are shown in Figure 7. We measure, for (i) the relative error defined as the fraction of the absolute error over the exact answer, and for (ii) and (iii) the average number of false positives and false negatives, i.e., number of time series that should not have been retrieved and number of time series that should have been retrieved but were not. For aggregate queries, relative error is large only for the MIN aggregate, since the PCA representation consistently overestimates the MIN[7]. For the selection queries, the number of both false positives and false negatives was small compared to the average query selectivity of 50.83. The results are equally good for the 20-NN queries. In fact PMC-MEAN had no false positives/negatives in this case.

5.2. Prediction Experiments

In our first experiment, we want to motivate experimentally the need for appropriate model selection as hinted in Section 4.2.2. We consider the location of an object moving in one dimension. This can be captured by a sensor, either on the object (e.g., GPS) or independent of it (e.g., radar). The object may move at a constant speed v for some length of time or accelerate/decelerate We generated 100 time series of length 500 for each type of motion, choosing $v \sim U(0, 50)$ and $a \sim U(0, 10)$. We added some measurement error $\sim U(-25, 25)$ on the location and tried to predict the location as: (i) last known location, (ii) first-order model (constant speed), (iii) second-order model (constant acceleration). We fit these models on the 10 most recent samples at prediction time. In Figure 8 we show the relative performance (number of parameter refreshes) using these three models, for varying ϵ_{pred} ranging from 10 to 160 meters. Not surprisingly, the best predictive model in each case is the one which generates the behavior. As an example, for the constant velocity series, the last-know-value is "too simple" failing to capture the change in the object's location, while the constant acceleration model is "too complex" and, despite encompassing the constant velocity model as a special case, fails to outperform it. Our example illustrates the benefit of pushing some intelligent behavior to the sensor and the importance of choosing a model carefully.

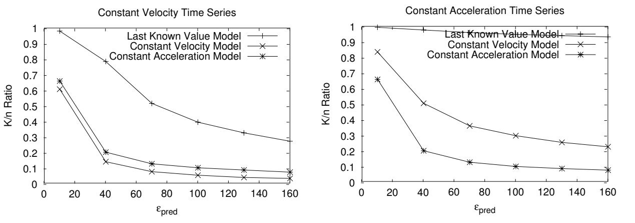

Figure 8. Model Selection

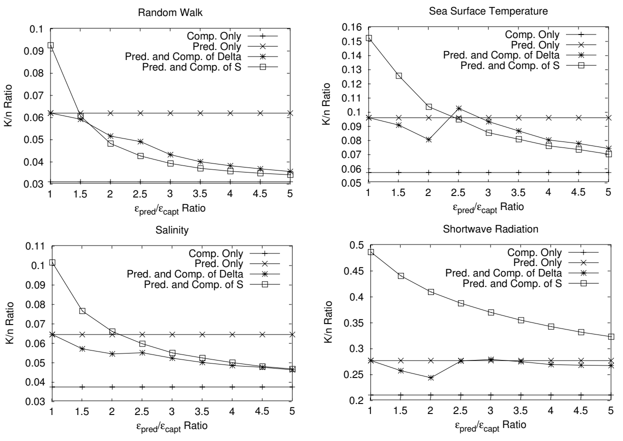

Figure 9. Prediction and Combined Prediction/Compression Experiments

[7]A lower bound on the MIN can be easily found, given that the PCA representation has an ϵ_{capt} guarantee

In our next experiment, we used a simple predictive model, namely predicting future values of the series as equal to its value at prediction time. This is optimal if the series is undergoing an unbiased random walk, since the expected value of the series at every future time is equal to its value at prediction time. Using this, each parameter update consists of a value and the prediction time. It thus has the same size as a segment of the PCA representation.

We use the same time series as before. We set $\epsilon_{capt} = 10\epsilon_{base}$, i.e., the "middle" value of our compression experiments. We simulate for ϵ_{pred} in 1- to 5-fold multiples of ϵ_{capt}. To conserve space, we combine a number of curves in the graphs of Figure 9. "Compression Only" is the number of segments for PMC-MR compression of S at ϵ_{capt} tolerance and "Prediction Only" is the number of parameter refreshes when $\epsilon_{pred} = \epsilon_{capt}$, i.e., when prediction alone is used to capture the time series. As we expect, compression works much better because it compresses values already seen optimally, rather than predicting future (uncertain) values. "Prediction and Compression of S" is the sum of a within-ϵ_{capt} compression of S and the number of parameter refreshes for ϵ_{pred} ranging from one to five times ϵ_{capt}. For "Prediction and Compression of Delta" we use of the result of Section 4.4 and compress the error series rather than S. As mentioned, when ϵ_{pred} is small, compressing Δ as opposed to S is preferrable. As ϵ_{pred} increases, the two curves approach each other. In the first two time series, compression of S is slightly preferrable, while in the other two the situation is reversed.

6. Related Work

Olston et al. [22] studies the performance/accuracy tradeoff with approximately replicated data. The motivation is in reducing communication, quantified as the number of exchanged messages between producer and the receiver of data. Interval-based approximations are stored at the receiver end, supplied as guarantees by the producer. Our work differs in employing a general model of approximate replication which considers temporal latency and combines compression and prediction. [22] proposes an algorithm for adaptively setting the interval width; this can also be used to adaptively set ϵ_{pred}. The adaptation problem was also studied by Deolasee et al. [10] for web data.

Sensor databases have recently been the center of much research in the database community e.g., in the Cougar [3] and Telegraph [20] projects. These efforts aim to create technology that will enable the creation of databases where sensors can be accommodated, taking into account the novel performance and semantic issues that distinguish sensors from traditional data sources.

Time series data has long been an important area of research. Our paper is not focused in introducing algorithms for extracting information from time series or in similarity retrieval as e.g., in Keogh et al. [17], or Agrawal et al. [1]. Our focus is in capturing sensor-generated series; applications similar to the above can then be applied to such series in the archive.

Chen et al. [7] propose compression of databases, motivated by the storage and bandwidth limitations of mobile devices. Unlike our paper, devices are the destinations of data. In Chen et al. [6] the problem of database compression and querying over compressed databases is studied. The authors motivate their work by the increase in CPU power, making it attractive to spend CPU time in compressing/decompressing data rather than in doing disk I/O for them. Our motivation is similar, making using sensors' CPU power to limit communication and energy drain.

In our paper, we use prediction as a means of improving system performance, namely saving communication and energy drain. This is different from the common use of prediction in which only the predicted values *themselves* are of interested. Gao et al. [12] also proposed to use prediction of time series values. In [12], the goal is to enable similarity-based pattern queries in batch mode by finding nearest neighbors of an incoming time series. By applying prediction on this time series, the system can generate candidate nearest neighbors ahead of time. When the actual values of the incoming series arrive, these are filtered and the actual nearest neighbors are returned.

Chen et al. [5] propose to perform on-line regression analysis over time series data streams. We also propose to fit models to time series, but our motivation is to improve system performance, rather than regression analysis. A useful extension to our work would be to use some of the ideas in [5] to address correlations between multiple time series that a single sensor may be monitoring.

Finally, we refer to work in moving object databases [28, 25, 19]. In this research field, we find the idea of approximating the time series of an object's location without continuous updates, in Wolfson et al. [28], of predicting an object's future location based on its velocity vector in Saltenis et al. [25], and of using the predictability of motion for improving performance in Lazaridis et al. [19].

7. Conclusions

In this paper we motivate the importance of capturing time series generated by wireless sensors. To achieve this we task sensors with compressing time series and fitting predictive models. We propose an optimal online algorithm for creating the piecewise-constant approximation of a real-valued time series, satisfying a bound on the L_∞ distance and show how prediction and compression can co-exist in a system to address the needs of both the time series capture task and real-time applications.

In the future, we plan to (i) evaluate the effectiveness of our techniques in a real-world setting, especially for motion time series, (ii) to examine how to evaluate general SQL queries with answer quality or response deadline tolerances, (iii) to develop adaptive algorithms for predictive model selection, and (iv) to investigate *lateral* communication between sensors, exploiting redundancy of information across many sensors to further improve performance in the time series capture setting.

Acknowledgements

Our work was supported by the National Science Foundation (Awards IIS-9996140, IIS-0086124, CCR-0220069, IIS-0083489) and by the United States Air Force (Award F33615-01-C-1902).

References

[1] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, 1993. Springer Verlag.

[2] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Symposium on Principles of Database Systems (PODS)*, 2002.

[3] P. Bonnet, J. Gehrke, and P. Seshadri. Towards sensor database systems. In *Mobile Data Management (MDM)*, 2001.

[4] K. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *ICDE Conference*, pages 126–133, 1999.

[5] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang. Multi-dimensional regression analysis of time-series data streams. In *VLDB Conference*, 2002.

[6] Z. Chen, J. Gehrke, and F. Korn. Query optimization in compressed database systems. In *ACM SIGMOD Conference*, 2001.

[7] Z. Chen and P. Seshadri. An algebraic compression framework for query results. In *International Conference on Data Engineering (ICDE)*, 2000.

[8] J. Cho and H. Garcia-Molina. Synchronizing a database to improve freshness. In *ACM SIGMOD Conference*, 2000.

[9] W. S. Conner, L. Krishnamurthy, and R. Want. Making everyday life easier using dense sensor networks. In *Ubicomp*, Lecture Notes in Computer Science. Springer, 2001.

[10] P. Deolasee, A. Katkar, A. Panchbudhe, K. Ramamritham, and P. Shenoy. Adaptive push-pull: disseminating dynamic web data. In *The tenth international World Wide Web conference on World Wide Web*, pages 265–274. ACM Press, 2001.

[11] J. Elson and D. Estrin. Time synchronization for wireless sensor networks. In *2001 International Parallel and Distributed Processing Symposium (IPDPS)*, 2001.

[12] L. Gao and X. S. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *ACM SIGMOD Conference*, 2002.

[13] W. Gilchrist. *Statistical Forecasting*. John Wiley & Sons, London, 1976.

[14] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser. Distributed sensor network for real time tracking. In *Proceedings of the fifth international conference on Autonomous agents*, pages 417–424. ACM Press, 2001.

[15] C. Hughes, J. Srinivasan, and S. Adve. Saving energy with architectural and frequency adaptations for multimedia applications. In *Proceedings of the 34th Annual International Symposium on Microarchitecture (MICRO-34), Dec. 2001.*, 2001.

[16] C. Intanagonwiwat, R. Govindan, and D. Estrin. Directed diffusion: a scalable and robust communication paradigm for sensor networks. In *Proceedings of the sixth annual international conference on Mobile computing and networking*, pages 56–67. ACM Press, 2000.

[17] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *ACM SIGMOD Conference*, 2001.

[18] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *International Conference on Data Mining*. IEEE Computer Society, 2001.

[19] I. Lazaridis, K. Porkaew, and S. Mehrotra. Dynamic queries over mobile objects. In *EDBT Conference*, 2002.

[20] S. Madden and M. J. Franklin. Fjording the stream: An architecture for queries over streaming sensor data. In *International Conference on Data Engineering (ICDE)*, 2002.

[21] M. J. McPhaden. Tropical atmosphere ocean project, pacific marine environmental laboratory. http://www.pmel.noaa.gov/tao/.

[22] C. Olston, B. T. Loo, and J. Widom. Adaptive precision setting for cached approximate values. In *ACM SIGMOD Conference*, 2001.

[23] C. Olston and J. Widom. Best-effort cache synchronization with source cooperation. In *ACM SIGMOD Conference*, 2002.

[24] G. J. Pottie and W. J. Kaiser. Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58, 2000.

[25] S. Saltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *SIGMOD Conference*, 2000.

[26] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *ICDE*, pages 536–545, 1996.

[27] L. Telksnys, editor. *Detection of changes in random processes*. New York, Optimization Software, 1986.

[28] O. Wolfson, S. Chamberlain, S. Dao, L. Jiang, and G. Mendez. Cost and imprecision in modeling the position of moving objects. In *ICDE Conference*, 1998.

[29] X. Yang and A. Bouguettaya. Broadcast-based data access in wireless environments. In *International Conference on Extending Database Technology (EDBT)*, 2002.

[30] S. Zdonik, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and D. Carney. Monitoring streams - a new class of data management applications. In *VLDB Conference*, 2002.