Declarative Support for Sensor Data Cleaning

Shawn R. Jeffery[1⋆], Gustavo Alonso[2⋆⋆], Michael J. Franklin[1], Wei Hong[3⋆], Jennifer Widom[4]

[1] UC Berkeley
[2] ETH Zurich
[3] Arched Rock Corporation
[4] Stanford University

Abstract. Pervasive applications rely on data captured from the physical world through sensor devices. Data provided by these devices, however, tend to be unreliable. The data must, therefore, be cleaned before an application can make use of them, leading to additional complexity for application development and deployment. Here we present *Extensible Sensor stream Processing* (*ESP*), a framework for building sensor data cleaning infrastructures for use in pervasive applications. ESP is designed as a pipeline using declarative cleaning mechanisms based on spatial and temporal characteristics of sensor data. We demonstrate ESP's effectiveness and ease of use through three real-world scenarios.

1 Introduction

Many pervasive applications rely on data collected from physical sensor devices such as wireless sensor networks and RFID technology. For instance, consider a sensor-enabled library (shown in Fig. 1) that uses RFID readers for detecting tags placed on books and patron's library cards, wireless sensors for monitoring environmental conditions, and various other devices such as motion and pressure sensors. Library monitoring and support applications use readings from these devices to manage inventory and checkouts, adjust temperature, and monitor patron activity. One of the main challenges in this scenario is the unreliability of the data produced by the sensor devices. These "dirty data" exist in two general forms:

- Missed readings: Sensors often employ low cost, low power hardware and wireless communication, which lead to frequently dropped messages. For example, RFID readers often capture only 60-70% of the tags in their vicinity [19]. Wireless sensors also demonstrate similar errors. For instance, in a wireless sensor network experiment at the Intel Research Lab in Berkeley, each sensor delivered, on average, only 42% of the data it was asked to report [24].
- Unreliable readings: Often, individual sensor readings are imprecise or unreliable. For instance, physical devices tend to "fail dirty": the sensor fails but continues to report faulty values. In a sensor network deployment in Sonoma County, CA, for example, 8 out of 33 temperature-sensing motes failed, but continued to report readings that slowly rose to above 100^o Celsius [34].

⋆ This work was done while the author was at Intel Research Berkeley.
⋆⋆ This work was done while the author was at UC Berkeley as a Stonebraker Fellow.

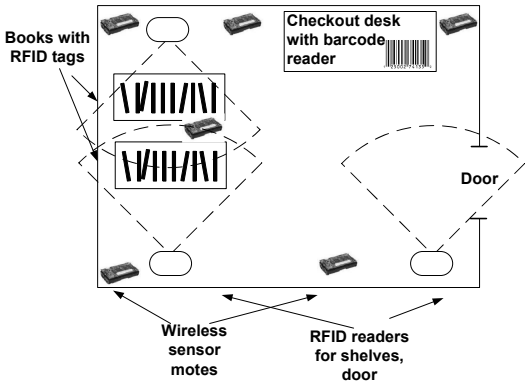
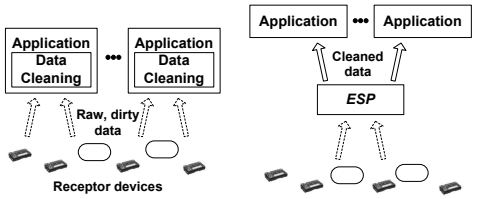


Fig. 1: A sensor-based library



(a) Current sensor-based deployments

(b) Using ESP as a data cleaning infrastructure

Fig. 2: An infrastructural approach to sensor data cleaning

To mitigate the effects of these errors, the data must be appropriately cleaned before use in an application. Of course, existing pervasive applications necessarily deal with these problems to some extent, but they tend to use tedious post-processing and application-specific means to clean sensor data (as shown in Fig. 2(a)). This ad-hoc treatment of unreliable data leads to brittle applications and increased deployment costs.

In contrast, we propose to separate cleaning from application logic by interposing a data cleaning infrastructure between sensor devices and applications (Fig. 2(b)). In such an approach, the cleaning infrastructure translates raw sensor data to cleaned data; applications are unaffected by the unreliable devices over which they are deployed.

In this paper, we present a framework for building cleaning infrastructures to support pervasive applications. *Extensible Sensor stream Processing* (or *ESP*), consists of a programmable pipeline of cleaning stages intended to operate on-the-fly as sensor data are streamed through the system. ESP is designed to be easy to configure and be able to evolve over time.

To provide a simple and flexible means of programming cleaning infrastructures, ESP uses declarative processing and exploits recent advances in relational processing techniques for data streams [4, 9, 12]. Programmers specify cleaning stages in ESP using high-level declarative queries over relational data streams[5]; the system then translates the queries into the appropriate low-level operations necessary to produce their results. Thus, programmers do not have to write low-level device interaction code (e.g., nesC for TinyOS [22]). Additionally, declarative languages provide data independence, such that in many cases cleaning operations do not need to be changed when devices fail, are added, or are upgraded. As an example of a declarative query for data cleaning, consider Query 1 which fills in lost temperature readings from a wireless sensor network using a 5 second moving average over each sensor's readings.

ESP utilizes the temporal and spatial nature of sensor data to drive many of its cleaning processes. Sensor data tend to be correlated in both time and space; the readings observed at one time instant are indicative of the readings observed at the next time

[5] In ESP, we use CQL [8] as our declarative language as we have a data stream system, TelegraphCQ [12], designed to process CQL. In principle, any declarative language would provide the benefits outlined here.

Query 1 *Example declarative query to interpolate for lost sensor readings. This query runs a 5 second moving average over each sensor's readings.*

```
SELECT node_id, avg(temperature)
FROM sensor_readings_stream [Range '5 sec']
GROUP BY node_id
```

instant, as are readings at nearby devices. Thus, we introduce the concepts of *temporal* and *spatial granule* to capture these correlations. These granules define a unit of time and space inside which the data are mostly homogeneous. These abstractions can be used to recover lost readings or remove outliers using temporal and spatial aggregation.

The ESP framework segments the cleaning process into five programmable stages, each responsible for a different logical aspect of the data, ranging from operations on individual readings to operations involving complex processing across multiple devices and outside data sources.

Of course, many applications need more advanced processing than that afforded by the declarative approach of ESP. We discuss such advanced processing later in the paper. Nevertheless, as demonstrated in this paper, infrastructures built with ESP's declarative stages are capable of cleaning sensor data in a wide range of deployments.

2 Related Work

Data cleaning is widely recognized as a crucial step for enterprise data management in the context of data warehouses. In this domain, data cleaning occurs separately from any application using the data (e.g., analytic/data mining software). Such traditional data cleaning, however, tends to focus on a small set of well-defined tasks, including transformations, matchings, and duplicate elimination [32, 23]. Extensions to this paradigm include the AJAX tool [21], an extensible, declarative means of specifying cleaning operations in a data warehouse. These techniques focus on offline cleaning for use in data warehouses; the real-time nature of many pervasive applications, however, preclude such approaches. More fundamentally, the nature of the errors in sensor data is not easily corrected by traditional cleaning: such technology typically does not utilize the temporal or spatial aspects of data.

The unreliabilities of sensor data have been widely studied. Work from ETH Zurich recognizes the poor behavior of RFID technology [19]. Work from the Intel Research Lab in Seattle has characterized the performance and errors in RFID technology in order to better guide ubiquitous applications [18, 30]. Other sensor-based applications have encountered similar issues in regard to dirty sensor data [11, 14]. These projects, however, either do not address cleaning or incorporate cleaning logic directly into the application.

Other work has advocated an infrastructural approach to sensor data access and management, but has not directly addressed data cleaning. Several systems provide mechanisms for interacting with wireless sensor networks ([27, 10]). For example, TinyDB provides a declarative means of acquiring data from a sensor network. ALE (Application-Level Events) defines an interface for building RFID middleware [7]. ALE defines concepts similar to our temporal and spatial granules. The Context Toolkit advocates an architectural approach to hiding the details of sensor devices [16].

Various projects have developed techniques for cleaning and error correction for wireless sensor data (e.g., [17, 28]). The BBQ system uses models of sensor data to accurately and efficiently answer wireless sensor network queries with defined confidence intervals [15]. Other work uses regression applied to sensor networks for inference purposes [29]. These approaches usually involve building and maintaining complex models. ESP's declarative approach, in contrast, does not rely on complex models.

Finally, we note that ESP is part of the HiFi project [20]. HiFi is a distributed stream processing system designed to support large-scale sensor-based networks (termed "high fan-in" systems). ESP is intended to clean sensor data streams at the edge of the HiFi network. Previous work discussed some of the preliminary concepts and results presented in this paper [20, 25].

3 ESP's Declarative Sensor Data Cleaning Framework

In this section, we introduce *Extensible Sensor stream Processing* (*ESP*), our declarative pipelined framework for building sensor data cleaning infrastructures.

While building the initial version of HiFi [13], we confronted many of the issues associated with unreliable data produced by sensor devices. Most notably, the system was unable to function correctly using raw RFID data. Our solution was to use a rudimentary pipeline of ad-hoc queries we termed "CSAVA" [20], designed to run throughout HiFi to convert RFID data into application data.

ESP generalizes and extends the CSAVA pipeline with a focus on cleaning sensor data at the edge of the network. ESP enables infrastructures that clean raw physical sensor data by processing multiple sensor streams, exploiting the temporal and spatial aspects of sensor data, to produce a single, improved output stream that can be used directly by pervasive applications. We first define the temporal and spatial abstractions that drive many of ESP's cleaning mechanisms.

3.1 Temporal and Spatial Granules

ESP uses high-level abstractions called *temporal* and *spatial granules* to capture time and space in sensor-based applications. These granules define units of time and space inside which the data are expected to be homogeneous. ESP uses the granule concept to aggregate, sample, and detect outliers. These abstractions exploit the fact that many applications are not interested in individual readings or devices, but with higher-level data in time and space.

Temporal Granules Although many sensor devices can produce data at frequent intervals, applications are usually concerned with data from a larger time period, or *temporal granule*. For instance, an environmental monitoring application that builds models of micro-climates in a redwood tree needs readings at 5 minute intervals to capture variations in micro-climate [35]. Within a temporal granule, readings are expected to be largely homogeneous.

To support this notion of temporal granules, ESP uses *windowed* processing to group readings. A window defines a finite set of readings (in terms of an interval of

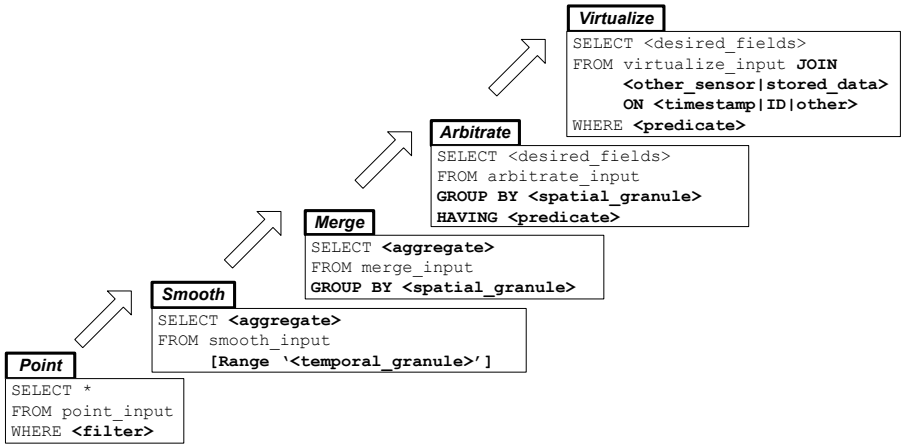


Fig. 3: ESP processing stages with the typical form of the declarative query for each stage. The relevant portion of the query is in boldface

time) within a data stream. Within a window, ESP can aggregate multiple readings into one or compare readings to detect outliers.

Spatial Granules Just as with readings in time, readings from devices physically close to each other are expected to be mostly homogeneous; a spatial granule defines the unit of space in which this homogeneity is expected to hold. Furthermore, a spatial granule is the smallest unit of space in which an application is interested, even though devices may have a finer spatial granularity. Examples of spatial granules include a shelf in a library scenario or a room in a digital home application.

To support spatial granules, ESP organizes sensors into *proximity groups*. A proximity group defines a set of sensors of the same type monitoring the same spatial granule. For instance, a set of motes monitoring the temperature in the same room may be grouped into the same proximity group, as may two RFID readers monitoring the same library shelf. Just as a time window is the unit of processing for a temporal granule, a proximity group is the processing unit for a spatial granule.

In many applications, the size of the temporal and spatial granules are obvious from the nature of the application or environment (e.g., 5 minute intervals in redwood monitoring or rooms in a digital home). In some cases, however, it may be desirable to determine the granule sizes automatically; this is a rich area of on-going work.

3.2 ESP Cleaning Stages

Having described the fundamental abstractions underlying ESP, we now outline ESP's processing stages. Through an analysis of typical sensor-based applications, we distilled a set of logically distinct operations that occur in a large class of applications to clean data produced by many types of sensor devices. Using these observations, ESP organizes sensor stream processing into a cascade of five programmable stages: *Point* - *Smooth* - *Merge* - *Arbitrate* - *Virtualize*. These stages operate on different aspects of the

data, from finest (single readings) to coarsest (readings from multiple sensors and other data sources). Not all stages are necessary for a given deployment.

Stage 1, *Point*: The *Point* stage operates over a single value in a sensor stream. The primary purpose of this stage is to filter individual values (e.g., errant RFID tags or obvious outliers) or to convert fields within an individual tuple. The general form for the *Point* query (as well as all other stages) is shown in Fig. 3. ESP applies the *Point* query to each sensor's readings, filtering all readings that do not match a predicate.

Stage 2, *Smooth*: In *Smooth*, ESP uses the temporal granule defined by the application to correct for missed readings and to detect outliers in a single sensor stream. The *Smooth* query processes its input stream, `smooth_input` (a stream of readings from a single device, provided by ESP), in windows of readings determined by the size of the temporal granule. For each of these windows, *Smooth* runs the specified aggregate function, outputs a processed reading, and then advances the window by one input reading. Note that both *Point* and *Smooth* operations can be pushed down to capable sensor devices (e.g., wireless motes).

Stage 3, *Merge*: Analogous to the temporal processing in the *Smooth* stage, *Merge* uses the application's spatial granule to correct for missed readings and remove outliers spatially. At each time step, *Merge* processes input readings from a single type of device and groups the readings by the specified spatial granule using the `GROUP BY` clause. *Merge* then processes each of these groups using an aggregate function to produce output readings for each spatial granule.

Stage 4, *Arbitrate*: Spatial granules may not map directly to sensor detection fields, leading to possible conflicts between the readings from different proximity groups that are physically close to one another. The *Arbitrate* stage deals with conflicts, such as duplicate readings, between data streams from different spatial granules. The query for *Arbitrate* groups its input stream by spatial granule and then uses the `HAVING` clause to filter readings from spatial granules that do not match a predicate.

Stage 5, *Virtualize*: Finally, some types of data cleaning utilize readings from across different types of sensors or stored data for improved data cleaning. To provide a platform for such techniques, the *Virtualize* stage combines readings from different types of devices and different spatial granules. The *Virtualize* query uses the `JOIN` construct to combine readings from different sources based on timestamps, IDs, or other common attributes. Additional processing can be specified using an optional predicate.

By separating sensor data cleaning into distinct stages, cleaning pipelines are easy to deploy and configure, affording many opportunities to reuse stages from previous deployments with changes localized to individual stages. Additionally, the cleaned data produced by ESP pipelines can be shared across many applications.

In the next three sections, we show detailed ESP processing and demonstrate ESP's overall effectiveness and ease of configuration with three typical sensor deployments.

4 RFID-based Scenario

The first deployment we address using ESP is a library scenario using RFID technology, similar to the one introduced in Sect. 1. RFID technology is notoriously error-prone:

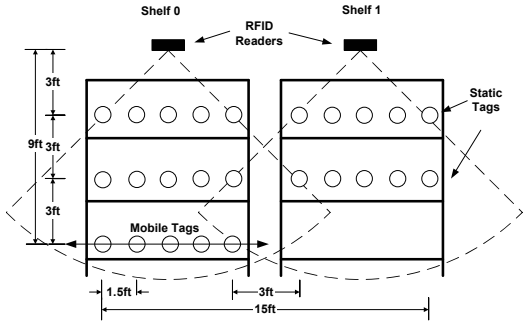


Fig. 4: Shelf scenario setup with 2 shelves, each with an RFID reader and 10 tags statically placed within 6 feet of the antenna (5 tags at 3 feet, 5 tags at 6 feet). Additionally, 5 tags were relocated every 40 seconds

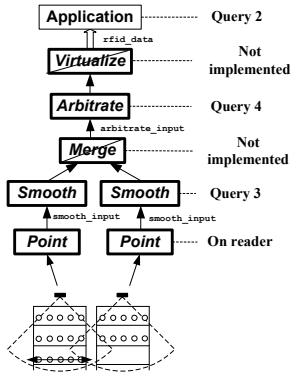
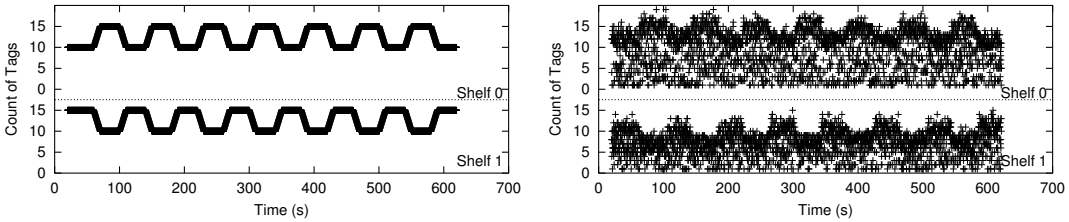


Fig. 5: ESP pipeline for cleaning RFID data



(a) Reality

(b) Query 2 results using raw RFID data

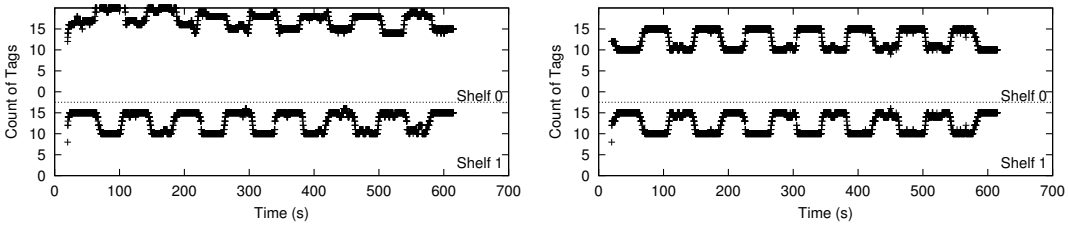
Fig. 6: Query 2 results in reality and over the raw data

tags that exist are frequently missed while other tags that are not in a reader's normal view are sometimes read. In a library scenario, consider an application that continuously monitors the count of books on each shelf using Query 2 (shown below). This query looks at the stream of RFID data in 5-second slices. Within each of these slices, the query groups the readings by the shelf at which the tag was read, and then counts the number of distinct tag IDs at each shelf. Here, the window clause indicates the temporal granule (5 seconds) and the `GROUP BY` clause denotes the spatial granule (a shelf).

Query 2 *Shelf monitoring query to determine the number of books on each shelf.*

```
SELECT shelf, count(distinct tag_id) as num_books
FROM rfid_data [Range '5 sec']
GROUP BY shelf
```

To study ESP used for cleaning RFID data, we ran an experiment emulating a library scenario. Our experimental setup is depicted in Fig. 4. We used two 915 MHz RFID readers from Alien Technology [6], each responsible for one shelf and thus each forming a proximity group. The readers' sample period was set at 5Hz (i.e., 5 polls per second). Each shelf was stocked with 10 books represented with Alien "I2" tags [5], EPC Class 1 RFID tags designed for long-range detection in a controlled environment. Tags were suspended in the same plane as the reader, spaced 1.5 feet apart from each other, and at two distances from the reader, 3 feet and 6 feet. Tags were oriented such



(a) Query 2 results after *Smooth* processing

(b) Query 2 results after *Arbitrate* processing

Fig. 7: Query 2 results after different stages of processing

that their antennae were directly facing the reader. Note that this setup is overly favorable to RFID technology as it attempts to alleviate many of the known causes of degraded readings [18, 19]. To introduce a dynamic component into the experiment, we relocated 5 tags placed 9 feet from the reader between the two shelves every 40 seconds.

The metric we use to evaluate our techniques is the average relative error of the results of Query 2, which is defined as $\frac{1}{N}\sum_{i=0}^{N}(\frac{|R_i - T_i|}{T_i})$, where N is total number of time steps, i is the time step at the granularity of the reader (5Hz), R_i is the reported count of tags on a shelf at time i, and T_i is the true count of tags on a shelf at i. This metric denotes how far off, on average, the reported count of tags is from reality. We ran this experiment three times; all runs produced similar results.

The results of our experiment without data cleaning are shown in Fig. 6. Figure 6(a) depicts the trace of the actual count of tags on each shelf over the course of the experiment. Figure 6(b) shows the results of running the application's query over the raw data. If the application were to use the output of the RFID readers directly, the results would be near-meaningless: the average relative error of the output of Query 2 compared to reality for the duration of the experiment was 0.41 (i.e., the count of the number of tags on each shelf was off by almost half, on average). For instance, if an application wants to be notified when the number of books on a shelf drops below 5, then the query using the raw data would report that a shelf has low inventory 2.3 times per second, on average.

We build a ESP pipeline to clean this data. Note that the RFID reader already provides *Point* functionality natively by removing tags that fail a checksum [1]. We use the *Smooth* and *Arbitrate* stages for ESP in this case (as shown in Fig. 5). As there is only one sensor per proximity group here, *Merge* is not needed.

4.1 Stage 2: *Smooth*

At the *Smooth* stage (shown in Query 3), ESP interpolates for lost readings within a temporal granule. ESP runs this query over each reader's data stream. This query begins by breaking the stream into 5-second slices (corresponding to the size of the temporal granule). For each of these slices, *Smooth* groups by tag ID and then counts the number of occurrences for that tag. The output of *Smooth*, then, is a reading for each tag seen at any point within the window and the number of times it was read. After each window is processed, ESP moves the window forward by one input reading. Through this sliding window operation, *Smooth* fills in dropped readings for any tag seen at least once in a 5 second time period.

Query 3 *Interpolating for lost readings in the* Smooth *stage.*

```
SELECT tag_id, count(*)
FROM smooth_input [Range '5 sec']
GROUP BY tag_id
```

The results of Query 2 over the data produced by this stage are shown in Fig. 7(a). The *Smooth* stage is able to eliminate the constant low inventory alerts generated by the query using the raw data.

The count of books per shelf, however, is still fairly inaccurate (an average relative error of 0.24) due to the close proximity of the readers and discrepancies in their performance. As seen in Fig. 7(a), the antenna for shelf 0 read more tags than that of shelf 1, despite being of the same model; the counts reported for shelf 0 were consistently 4 to 5 tags higher than reality. We tried different configurations of antennae and determined that this difference is likely due to known issues with the antenna ports on these particular RFID readers [2]. Processing in the *Smooth* stage has alleviated the issues with dropped readings, but any application using this data will be misled into thinking that shelf 0 has extra books.

4.2 Stage 4: *Arbitrate*

The *Arbitrate* stage (shown in Query 4) corrects for duplicate readings caused by the close proximity of the readers. At each time step, *Arbitrate* determines all tags that were read by multiple spatial granules and the number of times each tag was read by each granule. It then assigns the tag to the spatial granule that read the tag the most. ESP runs *Arbitrate* over the union of the streams produced by Query 3.[6]

Query 4 *Correcting for duplicate readings in the* Arbitrate *stage. The inner query determines the count of readings for a given tag in each spatial granule; the outer query selects the spatial granule with the highest count for each tag.*

```
SELECT spatial_granule, tag_id
FROM arbitrate_input ai1 [Range 'NOW']
GROUP BY spatial_granule, tag_id
HAVING count(*) >= ALL(SELECT count(*)
                       FROM arbitrate_input ai2
                       [Range 'NOW']
                       WHERE ai1.tag_id = ai2.tag_id
                       GROUP BY spatial_granule)
```

The results of running Query 2 over the smoothed and arbitrated data are shown in Fig. 7(b). Observe that ESP de-duplicates the readings as well as corrects for the differing performance of the two antennae to provide a substantially more accurate count of the tags on each shelf. After *Arbitrate* processing, the average relative error of Query 2 is 0.04. This equates to an error of being off by less than one book, on average. The results show that in this scenario, ESP provides a significant reduction in error over

[6] Although the *Merge* stage is unused in this case, ESP automatically adds a `spatial_granule` attribute to each stream, corresponding to each proximity group (i.e., each shelf).

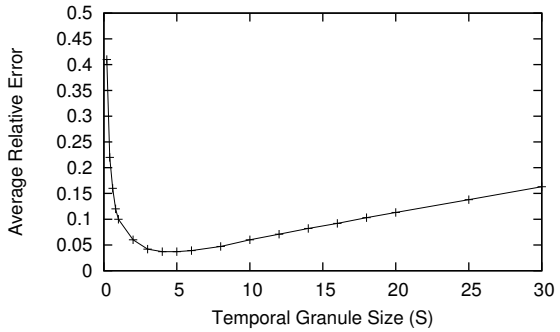


Fig. 8: Average relative error for Query 2 over data produced by ESP using different size temporal granules

the raw RFID data: recall that the original book counts using the raw data were off by almost half compared to reality.

Size of the Temporal Granule The size of the temporal granule affects the degree to which ESP can effectively clean the data. In order to effectively smooth, the size of the temporal granule (i.e., the window size) must be larger than the longest period of dropped readings in the input. The window size may not be made too large, however, as its size must be balanced with the rate of change of the data values. This tension can be observed in Figure 7(a), where the periods when tags are being relocated are not as accurately captured as the stable periods.

To investigate this issue, we compared the relative errors of ESP using different temporal granule sizes for the *Smooth* stage. The results are shown in Figure 8. At very small and very large granules, the error is larger than for granules around 5 seconds. Essentially, an effective temporal granule size is bounded at the low end by the reliability of the devices and at the high end by the rate of change of the data. In Sect. 7, we discuss our ongoing work exploring dynamic adaptation of the temporal granule.

5 Environment Monitoring Scenario

In the previous section, we demonstrated the ability of an ESP pipeline to clean RFID data streams. Next, we present a use case where ESP hides the unreliabilities of wireless sensor networks.

Wireless sensor networks enable new classes of pervasive applications that monitor environments such as the home and office with high resolution. In order to alleviate the effects of imprecise readings, calibration errors, outliers, and unreliable network communication, previous deployments involving sensor networks have had to post-process the readings, primarily by hand, to produce data that can be used by the application [11, 14, 15]. To reduce the complexity associated with sensor network application deployment, applications can use ESP to provide cleaned sensor data. We demonstrate two types of wireless sensor network data cleaning: outlier detection of fail-dirty motes, and temporal and spatial smoothing to correct for dropped messages.

5.1 Outlier Detection

Recall that sensor motes are known to "fail-dirty" and produce outlier readings. ESP can be used to alleviate the effects of these fail-dirty motes. To demonstrate the effectiveness of outlier detection using ESP, we use a 2 day trace from a sensor network deployed in the Intel Research Lab in Berkeley to monitor the lab's environment [24]. We focus on three motes in the same room, assigned to the same proximity group. In this trace, one of the motes fails by reporting increasing temperatures, rising to over 100^{o}C. We program the *Point* and *Merge* stages of ESP to eliminate the outlier readings. *Smooth* is not used because it cannot correct for extended errors produced by one sensor.[7] *Arbitrate* is not necessary as there is only one spatial granule.

Stage 1: *Point* The *Point* stage filters any readings beyond its expected range; in this case, ESP filters readings where the temperature is higher than 50^{o}C (Query 5) .

Query 5 *Simple filtering at the* Point *stage.*

```
SELECT *
FROM point_input
WHERE temperature < 50
```

Stage 3: *Merge* In this example, the *Merge* stage does outlier detection within a spatial granule by computing the average of the readings from different motes in the same proximity group and then omitting individual readings that are outside of two standard deviations from the mean (shown in Query 6). Note that these techniques are not intended to be statistically complex, but to the contrary, demonstrate the simplicity of ESP programming.

Query 6 *Outlier detection in the* Merge *stage.*

```
SELECT spatial_granule, AVG(temp)
FROM merge_input s [Range '5 min'],
     (SELECT spatial_granule, avg(temp) as avg,
                    stdev(temp) as stdev
      FROM merge_input [Range '5 min']) as a
WHERE a.spatial_granule = s.spatial_granule AND
      a.avg + (2*a.stdev) < s.temp AND
      a.avg - (2*a.stdev) > s.temp
```

Figure 9 shows the outcome of this experiment. The top line represents the outlier mote's readings. The middle line depicts the average of all three motes. If an application were to use the average of the three motes as a representation of the room's temperature, it would see temperatures exceeding 50^{o}C. The bottom lines show the traces of the two functioning motes as well as the output of ESP with outlier detection processing. Observe that ESP is able to detect when the outlier mote begins to deviate from the other motes and then omit its reading from its average calculation.

[7] *Smooth* could, however, be used to correct for individual outlier readings in a single mote using the same mechanisms presented here.

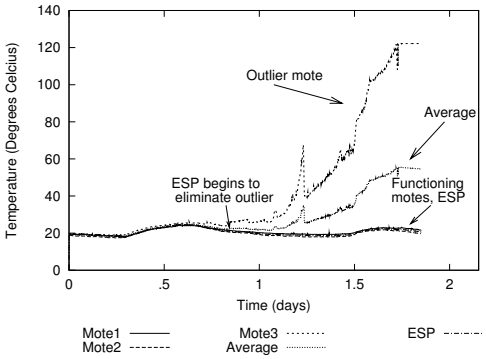


Fig. 9: Outlier Detection using ESP. The "ESP" line tracks the two functional motes' lines

5.2 Temporal and Spatial Smoothing of Sensor Data

Wireless sensor networks have another serious problem beyond fail-dirty motes: the network frequently drops messages. This problem is especially prevalent when sensor networks are deployed in the real world.

An ESP pipeline for a wireless sensor network can mask the unreliability of a sensor network by both temporally and spatially aggregating to correct for dropped readings. We demonstrate this cleaning through an application responsible for monitoring the temperature of a redwood tree at each elevation range in the tree.

We validated ESP processing on a three and a half day trace of data collected from sensors on a redwood tree in Sonoma County, CA as part of a large-scale sensor network deployment to study micro-climates of redwood trees [34]. 33 motes were placed along the trunk of the tree at varying heights. Data (e.g., temperature and humidity) were sensed at 5 minute intervals and logged to a local storage buffer (collected at the end of the experiment) and also sent over the multi-hop network. We grouped the motes at nearby heights into 2-node, non-overlapping proximity groups (corresponding to the spatial granules in this deployment), where the distance between motes in a proximity group was less than one foot.

Note that the log data is incorrect with respect to the ground truth due to fail-dirty sensors: 8 out of the 33 motes failed dirty. The readings from these motes were removed by hand shortly after data collection, but before we received the data.[8]

As ESP is addressing communication errors in this case, our metric of success is the epoch yield. Epoch yield describes the number of the readings reported to the application as a fraction of the total number of readings the application requested. For the raw data, the epoch yield in this trace was 40% (ideally, the epoch yield should be 100%). In other words, the application only received 40% of the data it requested. Additionally, we measure the percent error in the readings. Based on experience collaborating with biologists, an error of less than 1^oC is acceptable for trend analysis. Therefore, the goal of ESP in this application should be to increase the epoch yield while minimizing the percent of readings with an error greater than 1^oC.

Here, we implement the *Smooth* and *Merge* stages in ESP to temporally and spatially aggregate sensor readings to increase the epoch yield of the sensor deployment.

[8] ESP could employ the techniques shown in Section 5.1 to remove these outliers automatically.

Stage 2: *Smooth* In the *Smooth* stage (not shown), ESP temporally aggregates readings from a single sensor. By running a sliding window average on each sensor stream, lost readings from a single mote are masked within the window. After the *Smooth* stage, the epoch yield is increased to 77%. 99% of these readings were within 1^oC of the logged data.

Stage 3: *Merge* In the *Merge* stage (not shown), ESP performs spatial aggregation for each spatial granule (again, in the form of a windowed average) to further alleviate the effects of lost readings. The *Merge* stage increases the epoch yield to 92%. This improvement of reporting is at the slight cost of decreasing the percent of readings within 1^oC of the logged data to 94%. Thus, with ESP cleaning, biologists can get nearly complete data with a slight decrease in the accuracy.

Through the use of simple outlier detection and temporal and spatial smoothing, in this case an ESP pipeline is able to increase the ability of applications to make sense of the data they are getting from their sensors. Rather than spending time tediously post-processing the data, applications can focus on the high-level logic and not conversion, calibration, and error correction.

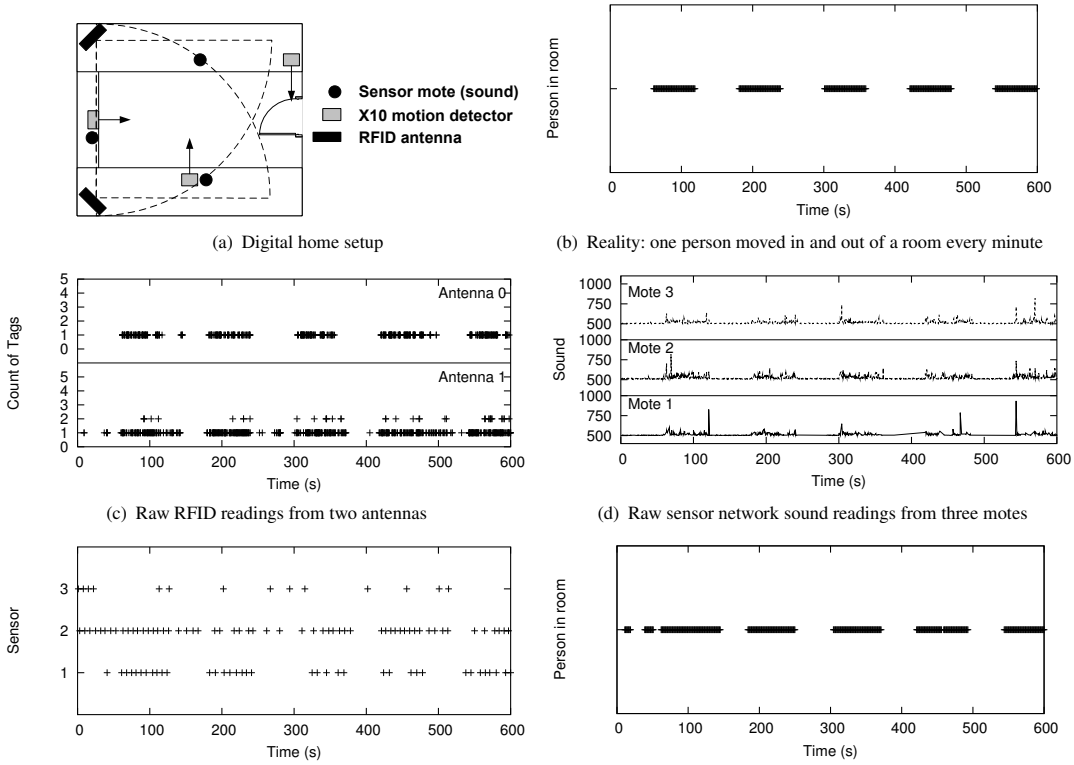
6 Digital Home Scenario

In Sects. 4 and 5, we demonstrated how ESP can provide a cleaning infrastructure to correct for a wide variety of problems associated with different physical devices. Next, we demonstrate the ease of configuration of ESP and highlight the use of multiple types of sensors to enhance data cleaning.

Multiple projects are developing sensors and infrastructures to instrument the home to provide both a better living experience for inhabitants as well as a more efficient use of home resources [3, 26]. Such applications use a wide variety of sensor devices providing low-level data (e.g., RFID, sensor motes, pressure sensors). In this section, we show that pipelines defined for other deployments (i.e., pipelines from the previous two sections) can be easily re-tasked to a new environment due to ESP's high-level declarative nature. Furthermore, ESP can serve platform for combining readings from multiple devices to provide a virtual "person detector" sensor. This type of processing is a higher level of cleaning; data from multiple heterogeneous devices, appropriately combined, can provide higher quality data. The output of ESP is a stream of events describing the presence of a person in the room.

We demonstrate the use of ESP in a digital home scenario by outfitting a room with two RFID readers, a small sensor network of three motes, and three X10 motion detectors [36] tasked to determine when someone is in the room (Fig. 10(a)). The room corresponds to one spatial granule for the application; thus, the two RFID readers make up one proximity group, the motes constitute another, and the X10 detectors form a third. During the experiment, one person, outfitted with an RFID tag, moved in and out of the room, while talking, at one minute intervals (Fig. 10(b)).

We present the ESP processing to clean the individual sensor streams and then describe how ESP utilizes these streams to create a person detector.



(a) Digital home setup

(b) Reality: one person moved in and out of a room every minute

(c) Raw RFID readings from two antennas

(d) Raw sensor network sound readings from three motes

(e) Raw X10 motion detectors. A mark indicates that the device reported movement

(f) Data after ESP processing

Fig. 10: A "Person Detector" in the digital home

6.1 Low-Level Sensor Cleaning

Recall the main advantages of ESP's declarative pipelined approach: previously built stages can be reused, changes necessary to tailor processing to each new deployment are isolated to small logical units, and these changes are easy to make and reason about as the stages are expressed as high-level queries. In this deployment, the programming for the ESP pipelines to clean the individual sensor streams (RFID, wireless sensors, and motion detectors) utilize almost exactly the same processing stages as defined in the previous two sections. Changes necessary for this deployment involved slightly modifying queries in a small number of stages. The raw data from these devices are presented in Figs. 10(c)- 10(e). We omit the details of this cleaning due to space considerations.

6.2 Stage 5: *Virtualize*

The main new feature of this use case (as compared to the previous scenarios) is the use of the *Virtualize* stage. *Virtualize* allows a deployment to combine readings from multiple different types of devices to perform application-level cleaning. In this case, *Virtualize* turns the set of heterogeneous devices into a "person detector." It uses a voting query that normalizes all sensor input streams to a single vote of whether it has

determined that a person is in the room or not (Query 7). The query then adds up the votes and registers that a person is in the room if the sum is higher than a threshold.

Query 7 *"Person Detector" logic at the* Virtualize *stage.*

```
SELECT 'Person-in-room'
FROM (SELECT 1 as cnt
      FROM sensors_input [Range 'NOW']
      WHERE sensors.noise > 525) as sensor_count,
     (SELECT 1 as cnt
      FROM rfid_input [Range 'NOW']
      HAVING count(distinct tag_id) > 1)
      as rfid_count,
     (SELECT 1 as cnt
      FROM motion_input [Range 'NOW']
      WHERE value = 'ON') as motion_count
WHERE sensor_count.cnt +
      rfid_count.cnt +
      motion_count.cnt >= threshold
```

The output of the ESP pipeline is shown in Fig. 10(f). As can be seen, simple and easy to deploy logic is capable of generally approximating reality. ESP is able to correctly indicate that a person is in the room 92% of the time.

***Virtualize* Configuration** The *Virtualize* query involves many numerical parameters, such as thresholds for sensor noise processing and overall voting. ESP's declarative query approach made this type of setup simple: high-level queries are easy to reason about and adjust until adequate cleaning is achieved. Furthermore, because ESP's cleaning is segmented, any adjustment of *Virtualize* is isolated to a single operation: lower-level cleaning remains the same. Nevertheless, there are many cases where this simple approach for *Virtualize* will not work; we discuss such cases in Sect. 7.

7 Advanced Cleaning

In Sects. 4, 5, and 6, we showed that cleaning infrastructures built using ESP are capable of cleaning data in a wide range of realistic scenarios. Perhaps surprisingly, these significant improvements in data quality were produced by a pipeline of fairly simple declarative queries. Of course, there are many applications and deployments where such a simple approach may not be effective. In this section, we outline some of these cases and discuss extensions to ESP that will enhance its effectiveness in such deployments.

Adaptive Granules In this paper, we required the application to supply the size of the temporal and spatial granules. In some cases, however, this is not possible: the environment may be too complex for the application to adequately determine appropriate sizes or too dynamic for a single size temporal or spatial granule to work. Thus, it is preferable to have the system determine and adapt the granule size based on the data it observes.

To this end, our on-going work involves modeling unreliable sensor data as a statistical sample of the physical world. With this model in place, we are investigating techniques from sampling theory to help guide data cleaning and granule size adaptations. For instance, ESP can use π-estimators [33] to determine the population of RFID tags in an area or the temperature of a set of sensors *without* seeing all the data. The variance of the estimator can be used to guide granule size decisions.

Soft Sensors While these deployments were configured using numerical parameters that were easy to derive empirically (e.g., *Virtualize* in Sect. 6), in many cases determining the parameter values may not be so easy. More advanced processing for *Virtualize* can involve machine learning techniques such as those used in soft sensors [31]. To support this type of operation, ESP can be extended to support stages defined by both declarative queries and user-supplied code.

Query-Driven Operations The cleaning infrastructures presented here have focused on providing "raw" (but cleaned) streams to the applications. In most cases, however, the application poses queries over these streams. Application-level queries are a mechanism for the application to alert the cleaning infrastructure of additional requirements. ESP should be able to incorporate this information to help drive cleaning operations. For instance, query predicates (e.g., `temp` $>$ `0`) should be pushed down to the appropriate level in the pipeline.

8 Conclusions

Data produced by physical sensor devices are notoriously dirty: readings are frequently either missed or dropped and individual readings are unreliable. Furthermore, these error characteristics vary from deployment to deployment. This leads to high application deployment costs for both data cleaning and configuration.

To directly address these issues, we developed ESP, a framework for building sensor data cleaning infrastructures in support of pervasive applications. By taking an infrastructural approach to sensor data cleaning, ESP allows applications to use sensor data without incorporating complex cleaning logic. Furthermore, applications using an ESP infrastructure can be write-once, run anywhere: ESP shields the application from changes in the error characteristics of the devices or the underlying environment. Finally, an infrastructure built using ESP allows multiple applications to use the same cleaned data, further reducing deployment costs.

To drive ESP's cleaning mechanisms, we introduce the concepts of temporal and spatial granules. These abstractions capture application-level notions of time and space. ESP utilizes these concepts in a pipeline of programmable processing stages designed to clean sensor data as it streams through the system.

ESP infrastructures are easy to deploy and evolve due to the following properties:

- Declarative: ESP cleaning logic is easy to program through high-level declarative queries. The system can utilize the well-understood techniques of relational query processing to efficiently execute these queries.

– Pipelined: ESP consists of separate, pipelined cleaning stages allowing operations to be independently programmed and reused across deployments.
– Cleaning framework: ESP defines logically distinct cleaning operations designed to directly address the error characteristics of sensor data.

While there are many complex operations that can be used to clean sensor data, we show here that in practice, some applications and deployments do not need such complexity. We validate the ESP platform through three real-world deployments demonstrating that infrastructures built using high-level declarative queries can successfully alleviate both missed and unreliable readings in sensor data. As a result, many pervasive applications were able to use data provided by ESP pipelines as they would any sensor data, but without many of the associated errors.

Sensor-based pervasive application development and deployment today is fraught with complexities stemming from the unreliable nature of devices on which they are built. Cleaning infrastructures built using ESP address these problems leading to reduced application complexity, faster deployment times with lower costs, and better manageability.

9 Acknowledgments

The authors would like to thank David Liu, Matt Denny, and other members of the Berkeley Database Group for feedback on early drafts, Nathan Burkhart for assisting in some of the experiments, Ryan Aipperspach for supplying insight and equipment in regards to pervasive computing, and the anonymous reviewers for providing many useful comments. This work was funded in part by NSF under ITR grants IIS-0086057 and SI-0122599, and by research funds from Intel and the UC MICRO program. G. Alonso was supported (in part) by the NCCR-MICS, a center supported by the Swiss National Science Foundation under grant number 5005-67322.

References

[1] Alien Technology. Nanoscanner Reader User Guide.
[2] Alien Technology. Personal correspondence.
[3] MIT House_n. http://architecture.mit.edu/house_n/.
[4] D. Abadi, *et al.*. Aurora: a data stream management system. In *SIGMOD*. 2003.
[5] Alien ALL-9250 I2 RFID tag. http://www.alientechnology.com/products/rfid-tags.
[6] Alien ALR-9780 915 MHz RFID Reader. http://www.alientechnology.com/products/rfid-readers/alr9780.php.
[7] Application Level Event (ALE) Specification Version 1.0. Http://www.epcglobalinc.org/standards_technology/EPCglobal_ApplicationALE_Specification_v112-2005.pdf.
[8] A. Arasu, *et al.*. The CQL continuous query language: Semantic foundations and query execution. *VLDB Journal*, (To appear).
[9] B. Babcock, *et al.*. Models and issues in data stream systems. In *SIGMOD*. 2002.
[10] P. Bonnet, *et al.*. Towards sensor database systems. In *Proc. Mobile Data Management*, volume 1987 of *Lecture Notes in Computer Science*. Springer, Hong Kong, January 2001.
[11] P. Buonadonna, *et al.*. TASK: Sensor Network in a Box. In *EWSN*. 2005.

[12] S. Chandrasekaran, *et al.*. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *CIDR*. 2003.
[13] O. Cooper, *et al.*. HiFi: A Unified Architecture for High Fan-in Systems. In *VLDB*. 2004.
[14] Demand-response. http://dr.me.berkeley.edu/.
[15] A. Deshpande, *et al.*. Model-Driven Data Acquisition in Sensor Networks. In *VLDB Conference*. 2004.
[16] A. K. Dey. *Providing Architectural Support for Building Context-Aware Applications*. Ph.D. thesis, Georgia Institute of Technology, 2000.
[17] E. Elnahrawy *et al.*. Cleaning and querying noisy sensors. In *WSNA '03: Proceedings of the 2nd ACM international conference on Wireless sensor networks and applications*. 2003.
[18] K. P. Fishkin, *et al.*. I Sense a Disturbance in the Force: Unobtrusive Detection of Interactions with RFID-tagged Objects. In *Ubicomp*. 2004.
[19] C. Floerkemeier *et al.*. Issues with RFID usage in ubiquitous computing applications. In *Pervasive Computing: Second International Conference, PERVASIVE 2004*. 2004.
[20] M. J. Franklin, *et al.*. Design Considerations for High Fan-In Systems: The HiFi Approach. In *CIDR*. 2005.
[21] H. Galhardas, *et al.*. Declarative data cleaning: Language, model, and algorithms. In *VLDB*, pp. 371–380. 2001.
[22] D. Gay, *et al.*. The nesC language: A holistic approach to networked embedded systems. In *SIGPLAN*. 2003.
[23] Informatica. http://www.informatica.com/.
[24] Intel Lab Data. http://berkeley.intel-research.net/labdata/.
[25] S. R. Jeffery, *et al.*. A Pipelined Framework for Online Cleaning of Sensor Data Streams. In *ICDE*. 2006.
[26] C. D. Kidd, *et al.*. The Aware Home: A Living Laboratory for Ubiquitous Computing Research. In *Cooperative Buildings*, pp. 191–198. 1999.
[27] S. Madden, *et al.*. The Design of an Acquisitional Query Processor For Sensor Networks. In *SIGMOD*. 2003.
[28] S. Mukhopadhyay, *et al.*. Data aware, low cost error correction for wireless sensor networks. In *WCNC*. 2004.
[29] M. A. Paskin, *et al.*. A robust architecture for distributed inference in sensor networks. In *IPSN*. 2005.
[30] M. Philipose, *et al.*. Mapping and Localization with RFID Technology. Technical Report IRS-TR-03-014, Intel Research, December 2003.
[31] S. Qin. Neural networks for intelligent sensors and control — practical issues and some solutions. In *Neural Networks for Control*. 1996.
[32] E. Rahm *et al.*. Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13, 2000.
[33] C.-E. Särndal, *et al.*. *"Model Assisted Survey Sampling"*. Springer-Verlag New York, Inc. (Springer Series in Statistics), 1992.
[34] Sonoma Redwood Sensor Network Deployment. http://www.cs.berkeley.edu/~get/sonoma/.
[35] G. Tolle, *et al.*. A macrosope in the redwoods. In *SenSys*. 2005.
[36] X10. http://www.x10.com.