

PRIDE: A Data Abstraction Layer for Large-Scale 2-tier Sensor Networks

Woochul Kang
University of Virginia
Email: wk5f@virginia.edu

Sang H. Son
University of Virginia
Email: son@cs.virginia.edu

John A. Stankovic
University of Virginia
Email: stankovic@cs.virginia.edu

Abstract—It is a challenging task to provide timely access to global data from sensors in large-scale sensor network applications. Current data storage architectures for sensor networks have to make trade-offs between timeliness and scalability. PRIDE is a data abstraction layer for 2-tier sensor networks, which enables timely access to global data from the sensor tier to all participating nodes in the upper storage tier. The design of PRIDE is heavily influenced by collaborative real-time applications such as search-and-rescue tasks for high-rise building fires, in which multiple devices have to collect and manage data streams from massive sensors in cooperation. PRIDE achieves scalability, timeliness, and flexibility simultaneously for such applications by combining a model-driven full replication scheme and adaptive data quality control mechanism in the storage-tier. We show the viability of the proposed solution by implementing and evaluating it on a large-scale 2-tier sensor network testbed. The experiment results show that the model-driven replication provides the benefit of full replication in a scalable and controlled manner.

I. INTRODUCTION

Recent advances in sensor technology and wireless connectivity have paved the way for next generation real-time applications that are highly data-driven, where data represent real-world status. For many of these applications, data streams from sensors are managed and processed by application-specific devices such as PDAs, base stations, and micro servers. Further, as sensors are deployed in increasing numbers, a single device cannot handle all sensor streams due to their scale and geographic distribution. Often, a group of such devices need to collaborate to achieve a common goal. For instance, during a search-and-rescue task for a building fire, while PDAs carried by firefighters collect data from nearby sensors to check the dynamic status of the building, a team of such firefighters have to collaborate by sharing their locally collected real-time data with peer firefighters since each individual firefighter has only limited information from nearby sensors [1]. The building-wide situation assessment requires fusing data from all (or most of) firefighters.

As this scenario shows, lots of future real-time applications will interact with physical world via large numbers of underlying sensors. The data from the sensors will be managed by distributed devices in cooperation. These devices can be either stationary (e.g., base stations) or mobile (e.g., PDAs and smartphones). Sharing data, and allowing timely access to global data for each participating entity is mandatory for successful collaboration in such distributed real-time applications.

Data replication [2] has been a key technique that enables each participating entity to share data and obtain an understanding of the global status without the need for a central server. In particular, for distributed real-time applications, the data replication is essential to avoid unpredictable communication delays [3][4].

PRIDE (Predictive Replication In Distributed Embedded systems) is a data abstraction layer for devices performing collaborative real-time tasks. It is linked to an application(s) at each device, and provides transparent and timely access to global data from underlying sensors via a scalable and robust replication mechanism. Each participating device can transparently access the global data from all underlying sensors without noticing whether it is from local sensors, or from remote sensors, which are covered by peer devices. Since global data from all underlying sensors are available at each device, queries on global spatio-temporal data can be efficiently answered using local data access methods, e.g., B+ tree indexing, without further communication. Further, since all participating devices share the same set of data, any of them can be a primary device that manages a sensor. For example, when entities (either sensor nodes or devices) are mobile, any device that is close to a sensor node can be a primary storage node of the sensor node. This flexibility via decoupling the data source tier (sensors) from the storage tier is very important if we consider the highly dynamic nature of wireless sensor network applications.

Even with these advantages, the high overhead of replication limits its applicability [2]. Since potentially a vast number of sensor streams are involved, it is not generally possible to propagate every sensor measurement to all devices in the system. Moreover, the data arrival rate can be high and unpredictable. During critical situations, the data rates can significantly increase and exceed system capacity. If no corrective action is taken, queues will form and the latencies of queries will increase without bound. In the context of centralized systems, several intelligent resource allocation schemes have been proposed to dynamically control the high and unpredictable rate of sensor streams [5][6][7]. However, no work has been done in the context of distributed and replicated systems.

In this paper, we focus on providing a scalable and robust replication mechanism. The contributions of this paper are:

- 1) a model-driven scalable replication mechanism, which

- significantly reduces the overall communication and computation overheads,
- 2) a global snapshot management scheme for efficient support of spatial queries on global data,
 - 3) a control-theoretic quality-of-data management algorithm for robustness against unpredictable workload changes, and
 - 4) the implementation and evaluation of the proposed approach on a real device with realistic workloads.

To make the replication scalable, PRIDE provides a model-driven replication scheme, in which the models of sensor streams are replicated to peer storage nodes, instead of data themselves. Once a model for a sensor stream is replicated from a primary storage node of the sensor to peer nodes, the updates from the sensor are propagated to peer nodes only if the prediction from the current model is not accurate enough. Our evaluation in Section 5 shows that this model-driven approach makes PRIDE highly scalable by significantly reducing the communication/computation overheads. Moreover, the Kalman filter-based modeling technique in PRIDE is lightweight and highly adaptable because it dynamically adjusts its model parameters at run-time without training.

Spatial queries on global data are efficiently supported by taking snapshots from the models periodically. The snapshot is an up-to-date reflection of the monitored situation. Given this fresh snapshot, PRIDE supports a rich set of local data organization mechanisms such as B+ tree indexing to efficiently process spatial queries.

In PRIDE, the robustness against unpredictable workloads is achieved by dynamically adjusting the precision bounds at each node to maintain a proper level of system load, CPU utilization in particular. The coordination is made among the nodes such that relatively under-loaded nodes synchronize their precision bound with an relatively overloaded node. Using this coordination, we ensure that the congestion at the overloaded node is effectively resolved.

To show the viability of the proposed approach, we implemented a prototype of PRIDE on a large-scale testbed composed of Nokia N810 Internet tablets [8], a cluster computer, and a realistic sensor stream generator. We chose Nokia N810 since it represents emerging ubiquitous computing platforms such as PDAs, smartphones, and mobile computers, which will be expected to interact with ubiquitous sensors in the near future. Based on the prototype implementation, we investigated system performance attributes such as communication/computation loads, energy efficiency, and robustness. Our evaluation results demonstrate that PRIDE takes advantage of full replication in an efficient, highly robust and scalable manner.

The rest of this paper is organized as follows. Section 2 presents the overview of PRIDE. Section 3 presents the details of the model-driven replication. Section 4 discusses our prototype implementation, and Section 5 presents our experimental results. We present related work in Section 6 and conclusions in Section 7.

II. OVERVIEW OF PRIDE

A. System Model

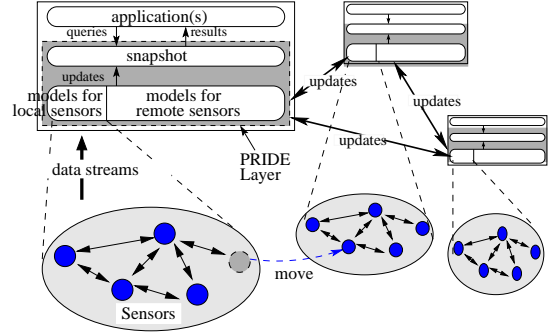


Fig. 1. A collaborative application on a 2-tier sensor network.

PRIDE envisions 2-tier sensor network systems with a *sensor tier* and a *storage tier* as shown in Figure 1. The sensor tier consists of a large number of cheap and simple sensors; $S = \{s_1, s_2, \dots, s_n\}$, where s_i is a sensor. Sensors are assumed to be highly constrained in resources, and perform only primitive functions such as sensing and multi-hop communication without local storage. Sensors stream data or events to a nearest storage node. These sensors can be either stationary or mobile; e.g., sensors attached to a firefighter are mobile.

The storage tier consists of more powerful devices such as PDAs, smartphones, and base stations; $D = \{d_1, d_2, \dots, d_m\}$, where d_i is a storage node. These devices are relatively resource-rich compared with sensor nodes. However, these devices also have limited resources in terms of processor cycles, memory, power, and bandwidth. Each storage node provides in-network storage for underlying sensors, and stores data from sensors in its vicinity. Each node supports multiple radios; an 802.11 radio to connect to a wireless mesh network and a 802.15.4 to communicate with underlying sensors. Each node in this tier can be either stationary (e.g., base stations), or mobile (e.g., smartphones and PDAs).

The sensor tier and the storage tier have loose coupling; the storage node, which a sensor belongs to, can be changed dynamically without coordination between the two tiers. This loose coupling is required in many sensor network applications if we consider the highly dynamic nature of such systems. For example, the mobility of sensors and storage nodes makes the system design very complex and inflexible if two tiers are tightly coupled; a complex group management and hand-off procedure is required to handle the mobility of entities [9].

Applications at each storage node are linked to the PRIDE layer. Applications issue queries to underlying PRIDE layer either autonomously, or by simply forwarding queries from external users. In the search-and-rescue task example, each storage node serves as both a in-network data storage for nearby sensors and a device to run autonomous real-time applications for the mission; the applications collect data by issuing queries and analyzing the situation to report results to the firefighter.

PRIDE is a data abstraction layer for the storage tier. It is linked to an application (or application infrastructure) and provides them efficient data access and query processing methods. Afterwards, a *node* refers to a *storage node* if it is not explicitly stated.

B. Usage Model

In PRIDE, all nodes in the storage tier are homogeneous in terms of their roles; no asymmetrical function is placed on a sub-group of the nodes. All or part of the nodes in the storage tier form a *replication group* R to share the data from underlying sensors, where $R \subset D$. Once a node joins the replication group, updates from its local sensors are propagated to peer nodes; conversely, the node can receive updates from remote sensors via peer nodes. Any storage node, which is receiving updates directly from a sensor, becomes a primary node for the sensor, and it broadcasts the updates from the sensor to peer nodes. However, it should be noted that, as will be shown in Section 3, the PRIDE layer at each node performs model-driven replication, instead of replicating sensor data, to make the replication efficient and scalable.

PRIDE is characterized by the queries that it supports. PRIDE supports both temporal queries on each individual sensor stream and spatial queries on current global data. Temporal queries on sensor s_i 's historical data can be answered using the model for s_i . An example of temporal query is "What is the value of sensor s_i 5 minutes ago?" For spatial queries, each storage node provides a snapshot on the entire set of underlying sensors (both local and remote sensors.) The snapshot is similar to a *view* in database systems. Using the snapshot, PRIDE provides traditional data organization and access methods for efficient spatial query processing. Currently, PRIDE support B+ tree, hashing, and queue data access methods. The access methods can be applied to any attributes, e.g., sensor value, sensor ID, and location; therefore, *value-based* queries can be efficiently supported. Basic operations on the access methods such as insertion, deletion, retrieval, and the iterating cursors are supported. Special operations such as *join cursors* for join operations are also supported by making indexes to multiple attributes, e.g., temperature and location attributes. This join operation is required to efficiently support complex spatial queries such as "Return the current temperatures of sensors located at room #4."

III. PRIDE DATA ABSTRACTION LAYER

The architecture of PRIDE is shown in Figure 2. PRIDE consists of three key components: (i) *filter & prediction engine*, which is responsible for sensor stream filtering, model update, and broadcasting of updates to peer nodes, (ii) *query processor*, which handles queries on spatial and temporal data by using a snapshot and temporal models, respectively, and (iii) *feedback controller*, which determines proper precision bounds of data for scalability and overload protection.

A. Filter & Prediction Engine

The goals of filter & prediction engine are to filter out updates from local sensors using models, and to synchronize

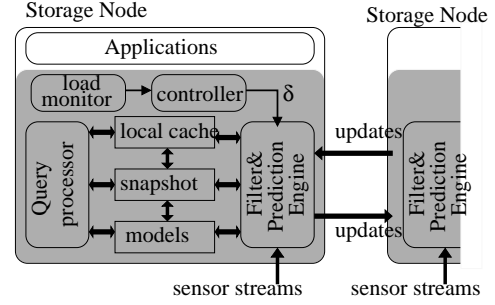


Fig. 2. The architecture of PRIDE (Gray boxes).

models at each storage node. The premise of using models is that the physical phenomena observed by sensors can be captured by models and a large amount of sensor data can be filtered out using the models. In PRIDE, when a sensor

Algorithm 1: OnUpdateFromSensor.

Input: update v from sensor s_i
1 \hat{v} =prediction from model for s_i ;
2 **if** $|\hat{v} - v| \geq \delta$ **then**
3 broadcast to peer storage nodes;
4 update data for s_i in the snapshot;
5 update model m_i for s_i ;
6 store to cache for later temporal query processing;
7 **else**
8 discard v (or store for logging);
9 **end**

Algorithm 2: OnUpdateFromPeer.

Input: update v from peer d_x
1 update data for s_x in the snapshot;
2 update model m_x for s_x ;
3 store to cache for later temporal query processing;

stream s_i is covered by PRIDE replication group R , each storage node in R maintains a model m_i for s_i . Therefore, all storage nodes in R maintain a same set of synchronized models, $M = \{m_1, m_2, \dots, m_n\}$, for all sensor streams in underlying sensor tier. Each model m_i for sensor s_i are synchronized at run-time by s_i 's current primary storage node (note that s_i 's primary node can change during run-time because of the network topology changes either at sensor tier or storage tier).

Algorithms 1 and 2 show the basic framework for model synchronization at a primary node and peer nodes, respectively. In Algorithm 1, when an update v is received from sensor s_i to its primary storage node d_j , the model m_i is looked up, and a prediction is made using m_i . If the gap between the predicted value from the model, \hat{v} , and the sensor update v is less than the precision bound δ (line 2), then the new data is discarded (or saved locally for logging.) This implies that the current models (both at the primary node and the peer nodes) are precise enough to predict the sensor output with the given precision bound. However, if the gap is bigger

than the precision bound, this implies that the model cannot capture the current behavior of the sensor output. In this case, m_i at the primary node is updated and v is broadcasted to all peer nodes (line 3). In Algorithm 2, as a reaction to the broadcast from d_j , each peer node receives a new update v and updates its own model m_i with v . The value v is stored in local caches at all nodes for later temporal query processing.

As shown in the Algorithms, the communication among nodes happens only when the model is not precise enough.

Models, Filtering, and Prediction So far, we have not discussed a specific modeling technique in PRIDE. Several distinctive requirements guide the choice of modeling technique in PRIDE. First, the computation and communication costs for model maintenance should be low since PRIDE handles a large number of sensors (and corresponding models for each sensor) with collaboration of multiple nodes. The cost of model maintenance linearly increases to the number of sensors. Second, the parameters of models should be obtained without an extensive learning process, because many collaborative real-time applications, e.g., a search-and-rescue task in a building fire, are short-term and deployed without previous monitoring history. A statistical model that needs extensive historical data for model training is less applicable even with their highly efficient filtering and prediction performance. Finally, the modeling should be general enough to be applied to a broad range of applications. Ad-hoc modeling techniques for a particular application cannot be generally used for other applications. Since PRIDE is a data abstraction layer for wide range of collaborative applications, the generality of modeling is important. To this end, we choose to use Kalman filter [10][6], which provides a systematic mechanism to estimate past, current, and future state of a system from noisy measurements. A short summary on Kalman filter follows.

Kalman Filter: The Kalman filter model assumes the true state at time k is evolved from the state at $(k - 1)$ according to

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{w}_k; \quad (1)$$

where

\mathbf{F}_k is the state transition matrix relating \mathbf{x}_{k-1} to \mathbf{x}_k ;

\mathbf{w}_k is the process noise, which follows $N(0, \mathbf{Q}_k)$;

At time k an observation \mathbf{z}_k of the true state \mathbf{x}_k is made according to

$$\mathbf{z}_k = \mathbf{H}_k \mathbf{x}_k + \mathbf{v}_k \quad (2)$$

where

\mathbf{H}_k is the observation model;

\mathbf{v}_k is the measurement noise, which follows $N(0, \mathbf{R}_k)$;

The Kalman filter is a recursive minimum mean-square error estimator. This means that only the estimated state from the previous time step and the current measurement are needed to compute the estimate for the current and future state.

In contrast to batch estimation techniques, no history of observations is required. In what follows, the notation $\hat{\mathbf{x}}_{n|m}$ represents the estimate of \mathbf{x} at time n given observations up to, and including time m . The state of a filter is defined by two variables:

$\hat{\mathbf{x}}_{k|k}$: the estimate of the state at time k given observations up to time k .

$\mathbf{P}_{k|k}$: the error covariance matrix (a measure of the estimated accuracy of the state estimate).

Kalman filter has two distinct phases: *Predict* and *Update*. The predict phase uses the state estimate from the previous timestep $k - 1$ to produce an estimate of the state at the next timestep k . In the update phase, measurement information at the current timestep k is used to refine this prediction to arrive at a new more accurate state estimate, again for the current timestep k . When a new measurement \mathbf{z}_k is available from a sensor, the true state of the sensor is estimated using the previous prediction $\hat{\mathbf{x}}_{k|k-1}$, and the weighted prediction error. The weight is called Kalman gain \mathbf{K}_k , and it is updated on each prediction/update cycle. The true state of the sensor is estimated as follows,

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k (\mathbf{z}_k - \mathbf{H}_k \hat{\mathbf{x}}_{k|k-1}). \quad (3)$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1}. \quad (4)$$

The Kalman gain \mathbf{K}_k is updated as follows,

$$\mathbf{K}_{k|k} = \mathbf{P}_{k|k-1} \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k)^{-1}. \quad (5)$$

At each prediction step, the next state of the sensor is predicted by,

$$\hat{\mathbf{x}}_{k+1|k} = \mathbf{F}_k \hat{\mathbf{x}}_{k|k-1}. \quad (6)$$

Example: For instance, a temperature sensor can be described by the linear state space, $\mathbf{x}_k = [x \quad \frac{dx}{dt}]^T$, where x is the temperature and $\frac{dx}{dt}$ is the derivative of the temperature with respect to time. As a new (noisy) measurement \mathbf{z}_k arrives from the sensor¹, the true state and model parameters are estimated by Equations 3 - 5. The future state of the sensor at $(k + 1)$ th time step after Δt can be predicted using the Equation 6, where the state transition matrix is

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix}. \quad (7)$$

It should be noted that the parameters for Kalman filter, e.g., \mathbf{K} and \mathbf{P} , do not have to be accurate in the beginning; they can be estimated at run-time and their accuracy improves gradually by having more sensor measurements. We do not need massive past data for modeling at deployment time. In addition, the *update* cycle of Kalman filter (Equations 3 - 5) is performed at all storage nodes when a new measurement is

¹Note that the temperature component of \mathbf{z}_k is directly acquired from the sensor, and $\frac{dx}{dt}$ can be indirectly calculated as the ratio of the temperature change to the elapsed time between the previous measurement and the current one.

broadcasted as shown in Algorithm 1 (line 5) and Algorithm 2 (line2). No further communication is required to synchronize the parameters of the models. Finally, as will be shown in Section 5, the *prediction/update* cycle of Kalman filter incurs insignificant overhead to the system.

B. Query Processor

The query processor of PRIDE supports both temporal queries and spatial queries with planned extension to support spatio-temporal queries.

Temporal Queries: Historical data for each sensor stream can be processed in any storage node by exploiting data at the local cache and linear smoother [10]. Unlike the estimation of current and future states using one Kalman filter, the optimized estimation of historical data (sometimes called *smoothing*) requires two Kalman filters, a forward filter \hat{x} and a backward filter \hat{x}_b . Smoothing is a non-real-time data processing scheme that uses all measurements between 0 and T to estimate the state of a system at a certain time t , where $0 \leq t \leq T$ (see Figure 3.) The smoothed estimate $\hat{x}(t|T)$ can be obtained as a linear combination of the two filters as follows.

$$\hat{x}(t|T) = \mathbf{A}\hat{x}(t) + \mathbf{A}'\hat{x}_b(t), \quad (8)$$

where \mathbf{A} and \mathbf{A}' are weighting matrices. For detailed discussion on smoothing techniques using Kalman filters, the reader is referred to [10].

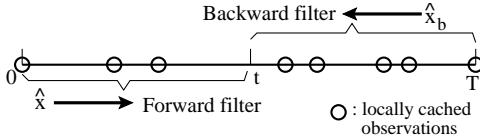


Fig. 3. Smoothing for temporal query processing.

Spatial Queries: Each storage node maintains a snapshot for all underlying local and remote sensors to handle queries on global spatial data. Each element (or data object) of the snapshot is an up-to-date value from the corresponding sensor. The snapshot is dynamically updated either by new measurements from sensors or by models². The Algorithm 1 (line 4) and Algorithm 2 (line 1) show the snapshot updates when a new observation is pushed from a local sensor and a peer node, respectively. As explained in the previous section, there is no communication among storage nodes when models well represent the current observations from sensors. When there is no update from peer nodes, the freshness of values in the snapshot deteriorate over time. To maintain the freshness of the snapshot even when there is no updates from peer nodes, each value in the snapshot is periodically updated by its local model. Each storage node can estimate the current state of sensor s_i using Equation 6 without communication to the primary storage node of s_i . For example, a temperature after 30 seconds can be predicted by setting Δt of transition matrix in Equation 7 to 30 seconds.

²Note that the data structures for the snapshot such as indexes are also updated when each value of the snapshot is updated.

The period of update of data object i for sensor s_i is determined, such that the precision bound δ is observed. Intuitively, when a sensor value changes rapidly, the data object should be updated more frequently to make the data object in the snapshot valid. In the example of Section 3.1.1, the period can be dynamically estimated as follows:

$$p[i] = \delta / \frac{dx}{dt}. \quad (9)$$

The $2 \times \delta / \frac{dx}{dt}$ is the *absolute validity interval (avi)* before the data object in the snapshot violates the precision bound, which is $\pm\delta$. The update period should be as short as the half of the *avi* to make the data object fresh [11].

Since each storage node has an up-to-date snapshot, spatial queries on global data from sensors can be efficiently handled using local data access methods (e.g., B+ tree) without incurring further communication delays.

C. Adaptive Data Quality Control

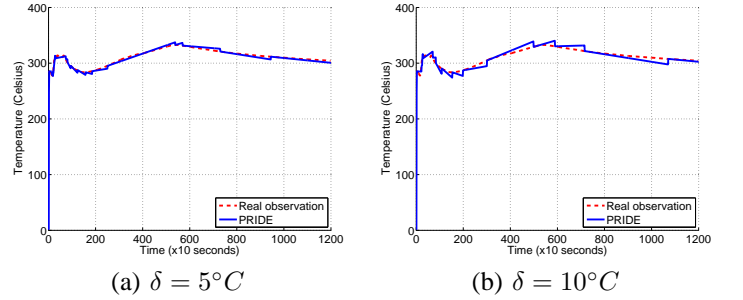


Fig. 4. Varying data precision.

Figure 4 shows how the value of one data object in the snapshot changes over time when we apply different precision bounds. As the precision bound is getting bigger, the gap between the real state of the sensor (dashed lines) and the current value at the snapshot (solid lines) increases. In the solid lines, the discontinued points are where the model prediction and the real measurement from the sensor are bigger than the precision bound, and subsequent communication is made among storage nodes for model synchronization. For applications and users, maintaining the smaller precision bound implies having a more accurate view on the monitored situation. However, the overhead also increases as we have the smaller precision bound.

Given the unpredictable data arrival rates and resource constraints, compromising the data quality for system survivability is unavoidable in many situations. In PRIDE, we consider processor cycles as the primary limited resource, and the resource allocation is performed to maintain the desired CPU utilization. The utilization control is used to enforce appropriate schedulable utilization bounds of applications can be guaranteed despite significant uncertainties in system workloads [12][5]. In utilization control, it is assumed that any cycles that are recovered as a result of control in PRIDE layer are used sensibly by the scheduler in the application layer to relieve the congestion, or to save power [12][5]. It can also enhance system survivability by providing overload protection against workload fluctuation.

Specification: At each node, the system specification $\langle U, \delta_{max} \rangle$ consists of a utilization specification U and the precision specification δ_{max} . The desired utilization $U \in [0..1]$ gives the required CPU utilization not to overload the system while satisfying the target system performance such as latency, and energy consumption. The precision specification δ_{max} denotes the maximum tolerable precision bound. Note there is no lower bound on the precision as in general users require a precision bound as short as possible (if the system is not overloaded.)

Local Feedback Control to Guarantee the System Specification: Using feedback control has shown to be very effective for a large class of computing systems that exhibit unpredictable workloads and model inaccuracies [13]. Therefore, to guarantee the system specification without a priori knowledge of the workload or accurate system model we apply feedback control.

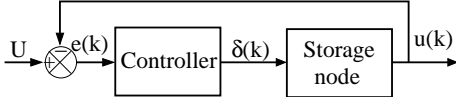


Fig. 5. The feedback control loop.

The overall feedback control loop at each storage node is shown in Figure 5. Let T is the sampling period. The utilization $u(k)$ is measured at each sampling instant $0T, 1T, 2T, \dots$ and the difference between the target utilization and $u(k)$ is fed into the controller. Using the difference, the controller computes a local precision bound $\delta(k)$ such that $u(k)$ converges to U .

The first step for local controller design is modeling the target system (storage node) by relating $\delta(k)$ to $u(k)$. We model the the relationship between $\delta(k)$ and $u(k)$ by using profiling and statistical methods [13]. Since $\delta(k)$ has higher impact on $u(k)$ as the size of the replication group increases, we need different models for different sizes of the group. We change the number of members of the replication group exponentially from 2 to 64 and have tuned a set of first order models $G_n(z)$, where $n \in \{2, 4, 8, 16, 32, 64\}$. $G_n(z)$ is the z-transform transfer function of the first-order models, in which n is the size of the replication group. After the modeling, we design a controller for the model. We have found that a proportional integral (PI) controller [13] is sufficient in terms of providing a zero steady-state error, i.e., a zero difference between $u(k)$ and the target utilization bound. Further, a gain scheduling technique [13] have been used to apply different controller gains for different size of replication groups. For instance, the gain for $G_{32}(z)$ is applied if the size of a replication group is bigger than 24 and less than or equal to 48. Due to space limitation we do not provide a full description of the design and tuning methods.

Coordination among Replication Group Members: If each node independently sets its own precision bound, the net precision bound of data becomes unpredictable. For example, at node d_j , the precision bounds for local sensor streams are determined by d_j itself while the precision bounds for remote

sensor streams are determined by their own primary storage nodes.

Algorithm 3: PrecisionBoundControl

```

Input: myid: my storage id number
1 /* Get local  $\delta$ . */
2 measure  $u(k)$  from monitor;
3 calculate  $\delta_{myid}(k)$  from local controller;
4 foreach peer node  $d$  in  $R - \{d_{myid}\}$  do
5     /* Exchange local  $\delta$ s. */
6     /* Use piggyback to save communication cost. */
7     send  $\delta_{myid}(k)$  to  $d$ ;
8     receive  $\delta_i(k)$  from  $d$ ;
9 end
10 /* Get the final global  $\delta$ . */
11  $\delta_{global}(k) = \max(\delta_i(k))$ , where  $i \in R$ ;

```

PRIDE takes a conservative approach in coordinating storage nodes in the group. As Algorithm 3 shows, the global precision bound for the k th period is determined by taking the maximum from the precision bounds of all nodes in the replication group. The global precision bound at k th sampling period, is the common precision bound for all storage nodes in the replication group for the period. We choose the greatest value among all precision bounds from each local control loop since a node that has the greatest precision bound is the most overloaded node in period k . By setting the global precision bound to the most overloaded node's, we can ensure that congestion at the most overloaded node is reduced.

IV. IMPLEMENTATION

PRIDE: PRIDE is an extension of Berkeley DB [14], which is a popular open-source embedded database. Unlike traditional database systems, Berkeley DB is embeddable to an application; Berkeley DB is linked to an application (or application infrastructure) and provides robust storage features such as diverse access methods, ACID transactions, recovery, locking, and multi-threading for concurrency. PRIDE exploits these features of Berkeley DB and extends them by providing model-driven replication, snapshot management, and dynamic data precision control. Snapshots of PRIDE are database files of Berkeley DB on which the diverse data management functionalities of Berkeley DB can be applied.

Testbed: We have implemented a prototype of PRIDE on a multi-tier sensor network testbed. In the testbed, the storage tier employs one Nokia N810 Internet tablet and a Centurion cluster machine. The N810 device is equipped with 400MHz TI OMAP processor, 128MB RAM, 256MB flash memory, 802.11b Wi-Fi radio, and runs the Linux 2.4.19 kernel. The Centurion cluster enables a large-scale evaluation. The cluster is equipped with 64 computing nodes, in which each node has two 1.5 GHz AMD Opteron processors, 2GB RAM. Each node of the Centurion cluster emulates one storage node. However, the real measurements of CPU utilization, energy consumption, and the query latency are performed in the N810 device. The N810 device and the Centurion cluster is connected via wireless Ethernet.

In the sensor tier, sensor streams can be generated by NIST CFAST (The Consolidated Model of Fire and Smoke Transport) fire simulator [15]. Using CFAST simulator, a wide-range of fire scenarios can be simulated in detail by configuring the input parameters, which include the geometry of the compartments and the connections between these compartments, the initial fire source and burning objects in the compartments, flow vents, and floor/wall materials. Traces are generated from the CFAST simulator in off-line for repeatability and scalability of experiments. For large scale experiments, additional sensor streams can be made by time-shifting the original traces. These traces are replayed and sent to storage nodes by another node of the Centurion cluster.

V. EVALUATION

In this section, we evaluate the performance of PRIDE in our testbed. For evaluation, a fire in a 10-story building with 100 compartments is modeled using the CFAST simulator, in which temperatures at 1024 locations are measured for 30 minutes with 1 second intervals. The PDAs carried by 32 firefighters are emulated by storage nodes in the testbed (one N810 device and 31 Centurion cluster nodes). On every 0.5 second period³, a new query is automatically issued by an application at each storage node. The queries randomly access 512 sensor values in the snapshot⁴.

A. Performance of Model-Driven Replication

First, we show the performance of the model-driven replication scheme in PRIDE. We compare the performance of PRIDE with three baseline algorithms.

- **Full (Full replication):** All updates from sensor streams are fully replicated to peer storage nodes in the replication group.
- **Approx-Caching (Approximate caching):** This is a value-driven approach, in which a storage node broadcasts updates from sensors only if the difference between current data and last broadcasted data is larger than a threshold (δ). This approach is similar to the algorithm in [7].
- **PRIDE/NU (No dynamic snapshot update):** This is the PRIDE approach, but the snapshot at each node is not dynamically updated by models. This approach is included to reveal the overhead of dynamic snapshot update using models in PRIDE.

The data quality controller in PRIDE and PRIDE/NU is turned off during the evaluation to assess the efficiency of the model-driven replication alone. The data precision bound δ is $1^\circ C$ for both PRIDE and Approx-Caching. All evaluation results are based on at least 5 runs and the averages with 95% confidence intervals are taken⁵.

³Real-time queries for firefighters can be invoked on a per-second basis [16].

⁴The applications can use raw data returned from the queries for further analysis and decision making. However, further processing in the application layer is not modeled in the evaluation.

⁵The confidence interval bars are shown in the graphs.

1) *Scalability:* Figure 6 shows the scalability of PRIDE and baselines when we change the number of storage nodes from 2 to 16. Each storage node receives data streams from 100 sensors. Hence, the number of underlying sensors increases from 200 to 1,600 accordingly. In Figure 6-(a), as the system size increases, the number of messages increases in all approaches. However, the slope is much flatter in PRIDE than baseline approaches since PRIDE filters out most of the incoming data from sensors as long as its models can predict the value within the precision bound. For instance, PRIDE filters out 93% of the original data when 16 storage nodes are deployed while Approx-Caching filters out only 80%. This gap increases as the system scales up. The high filtering rates of PRIDE implies that it can be highly robust and scalable in low-bandwidth networking environments.

The amount of communication is highly related to the CPU load since each message incurs overhead to handle it. Figure 6-(b) shows the CPU utilization in the same experiment. The CPU loads increase proportionally to the amount of communication in all approaches. Maintaining a proper level of CPU load is particularly important for real-time applications to guarantee the deadlines of real-time tasks. Once CPU load rises above a certain scheduling bound, the lengths of scheduling queues in the system increase limitlessly. In Figure 6-(c), the query latency increases without limit as CPU gets saturated in the Full approach. The relatively flat slope of PRIDE implies that the scale of PRIDE can be much higher than the baseline approaches before the system gets overloaded.

In PRIDE, CPU load is not only related to the number of exchanged messages, but also to the number of underlying sensors since PRIDE updates data objects in the snapshot periodically using models of each sensor. In Figure 6-(b), the gap between PRIDE and PRIDE/NU quantifies the cost of the dynamic snapshot update using models. PRIDE incurs less than 10% CPU overhead than PRIDE/NU when 16 storage nodes are involved. This implies that the CPU overhead to maintain models in PRIDE is insignificant.

Finally, note that we can reduce the number of communication messages (and subsequent resource consumption) dramatically by concatenating several individual messages. For example, the CPU loads decreased more than 70% in all approaches when messages for 1 second period were concatenated and bulk-transferred in one message. However, this benefit applies to all approaches and the relative performance remains the same. Further, this gain is acquired at the cost of additional delays in the message propagation. PRIDE supports this bulk-transfer option for applications having loose timing constraints.

2) *Energy Consumption:* Since PRIDE targets (potentially mobile) low-end devices such as PDAs, and micro servers, energy is a critical resource for long lifetime. In this section, we compare the energy consumption of PRIDE to the baseline approaches. In the evaluation, 16 storage nodes are deployed for 30 minutes, where each storage node handles 100 sensor streams. The power measurements are performed in the N810 device by monitoring the remaining battery power. The full

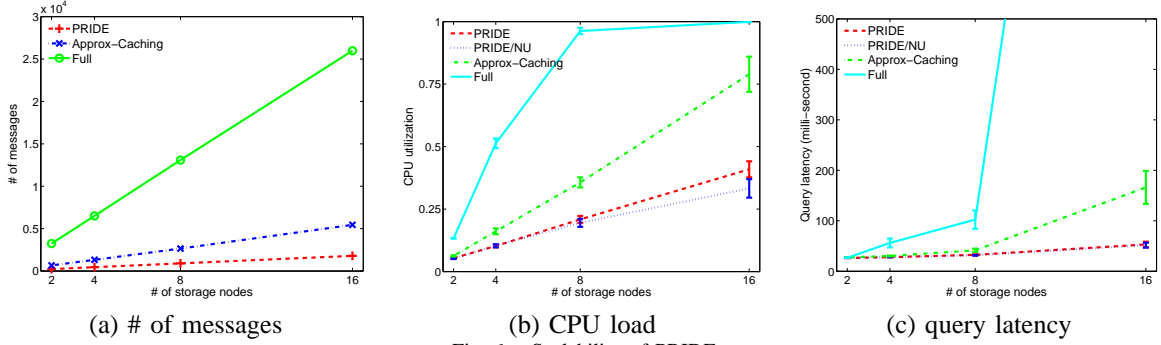


Fig. 6. Scalability of PRIDE.

capacity of the battery is 1,500mAh.

Figure 7-(a) shows the energy consumption after 30 minutes, and Figure 7-(b) shows the changes in the remaining battery power over 30 minutes. The result shows that PRIDE consumes the smallest energy in the experiment; PRIDE consumes 27% and 43% less energy than the Approx-Caching and the Full approach, respectively. This result is expected since the overhead in computation and communication has net effect on the energy consumption. Note that PRIDE consumes insignificantly small additional energy compared to PRIDE/NU, which does not perform the dynamic snapshot update. The additional energy consumption is less than 1.5%. This result again demonstrates that the benefit of using models for replication outperforms the cost of maintaining the models and snapshot.

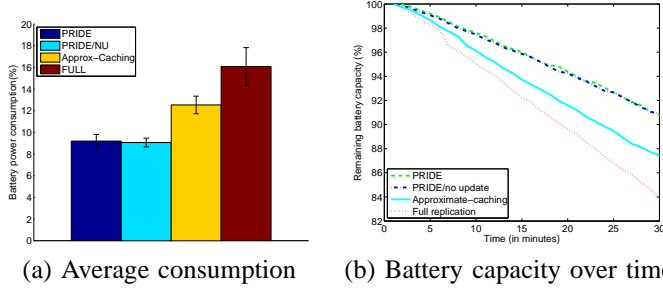


Fig. 7. Energy consumption.

B. Adaptability

We next evaluate the adaptability of PRIDE against unpredictable workloads. For the evaluation, 32 storage nodes are deployed and the data quality controller at each node is turned on for the experiment. The performance goal is given by the specification $\langle 0.6, 10^\circ C \rangle$, which means the CPU utilization bound is 60% and the maximum precision bound δ is $10^\circ C$. In the experiment, the sampling intervals of controllers are set to 20 seconds. We compare the performance when the data quality controllers are turned on and off.

1) *Average Performance*: We evaluate the adaptability of PRIDE by changing the workload. The workloads are varied by changing the number of sensor streams for each storage nodes from 60 to 140. For PRIDE without controllers, the precision bound is set to $3^\circ C$.

In Figure 8, the average performance is shown. Figure 8-(a) shows that PRIDE with controller achieves the target

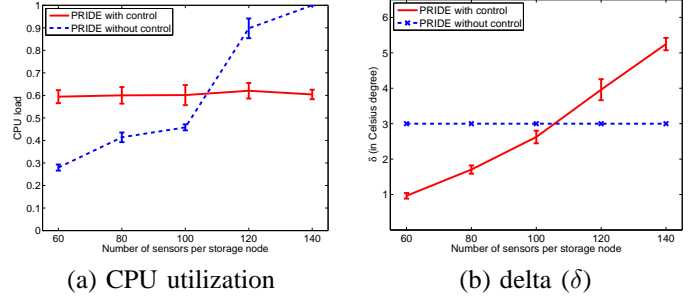


Fig. 8. Varying workload.

CPU load closely in all workloads. In contrast, the CPU load fluctuates dramatically between under-utilization and over-utilization when no control is applied; the load changes between 0.28 and 1. Violating the goal in CPU load implies that the latency of application tasks and queries in PRIDE can be increased significantly. Figure 8-(b) shows the changes in δ to achieve the target CPU load. In PRIDE, the precision bound δ increases linearly as the workload increases. However, PRIDE still satisfies the maximum precision bound, which is $10^\circ C$.

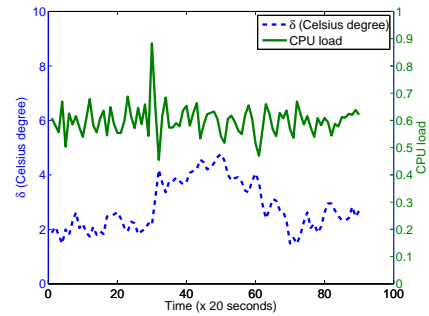


Fig. 9. Transient behavior.

2) *Transient Performance*: The average performance is not enough to show the performance of dynamic systems. Transient performance such as settling time should be small enough to satisfy the requirements of applications. In this experiment, the workload is increased by changing the number of underlying sensors and the transient behavior is observed. For example, we can consider a situation in which a firefighter moves to a location where sensors are densely deployed. The number of sensors at one of the storage nodes (the N810

device) is increased from 80 to 140 for 30 sampling periods.

Figure 9 shows that the CPU load and the precision bound δ . The increase of workload starts at the 30th sampling period. We can see that the CPU load increases suddenly at the 30th sampling period. However, the CPU load stabilizes within 2 sampling periods. When the workload decreases to the original level at the 60th sampling period, it takes 1 sampling period to stabilize.

VI. RELATED WORKS

Many data-centric approaches have been proposed to efficiently store and query data from sensors. It ranges from centralized approaches [17][18] to fully distributed storages [19][20]. However, both approaches manifest their own problems. For centralized approaches, the scale of the system is limited and their resource consumption can be unbalanced due to funnelling effect [21]. Conversely, the fully distributed approaches cannot provide timeliness in data accesses and query processing. TSAR [22] is a 2-tier data storage architecture that lies between the two extremes, in which the limitations of the both approaches are overcome by employing local archiving at the sensors and distributed indexing at the resource-rich proxies. However, TSAR still incurs communication delays at query processing time, which can be unacceptable for real-time applications such as aforementioned search-and-rescue tasks for a building fire. Moreover, the distributed indexing scheme in TSAR tightly couples the sensor tier and the upper proxy tier, limiting the flexibility of the system. In contrast, the model-driven replication mechanism in PRIDE provides timeliness, scalability, and flexibility simultaneously.

Recently, reducing the communication cost using filters and models have been an active research issue. In particular, the model-driven approaches are highly related to PRIDE. BBQ [23] exploits time-varying multivariate Gaussian and Kalman filter, and PRESTO [24] uses ARIMA prediction models to minimize the resource consumption. Jain and et.al. [6] introduced dual Kalman filters. PRIDE exploits the best results of these previous approaches. However, those approaches are different from PRIDE since they focus on the modeling and filtering in a centralized system. In contrast, PRIDE uses models to reduce the communication cost in distributed and replicated environments.

VII. CONCLUSIONS

This paper introduced the design, implementation, and experimental evaluation of a new data abstraction layer for 2-tier sensor network applications. PRIDE achieves the scalability, timeliness, and flexibility simultaneously by integrating model-driven full replication and adaptive data quality control in the storage tier. The model-driven replication in PRIDE provides the benefit of full replication in a highly scalable and controlled manner. In particular, the modeling technique in PRIDE has insignificant overhead and does not need a vast amount of past data for model training. We implemented PRIDE in a large-scale 2-tier sensor network testbed. Our experimental evaluation of PRIDE with realistic workloads

demonstrates the benefit and feasibility of model-driven full replication scheme in large-scale 2-tier sensor networks.

REFERENCES

- [1] K. Sha, W. Shi, and O. Watkins, "Using wireless sensor networks for fire rescue applications: Requirements and challenges," in *IEEE International Conference on Electro/information Technology*, 2006.
- [2] J. Gray, P. Helland, P. O'Neil, and D. Shasha, "The dangers of replication and a solution," in *SIGMOD '96*, 1996.
- [3] C. Huang, G. Zhou, T. F. Abdelzaher, S. H. Son, and J. A. Stankovic, "Load balancing in bounded-latency content distribution," in *RTSS '05*, 2005.
- [4] Y. Wei, S. H. Son, J. A. Stankovic, and K. D. Kang, "Qos management in replicated real time databases," in *RTSS '03: Proceedings of the 24th IEEE International Real-Time Systems Symposium*, 2003.
- [5] N. Tatbul, U. Çetintemel, S. Zdonik, M. Cherniack, and M. Stonebraker, "Load shedding in a data stream manager," in *VLDB '2003: Proceedings of the 29th international conference on Very large data bases*, 2003.
- [6] A. Jain, E. Y. Chang, and Y.-F. Wang, "Adaptive stream resource management using kalman filters," in *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. New York, NY, USA: ACM Press, 2004, pp. 11–22.
- [7] C. Olston, B. T. Loo, and J. Widom, "Adaptive precision setting for cached approximate values," *SIGMOD Rec.*, vol. 30, no. 2, pp. 355–366, 2001.
- [8] "Nokia N-Series, <http://www.nseries.com/>," 2008.
- [9] T. Abdelzaher and et. al., "Envirotrack: Towards an environmental computing paradigm for distributed sensor networks," in *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems*, 2004.
- [10] A. Gelb, Ed., *Applied Optimal Estimation*. The M.I.T Press, 1974.
- [11] K. Ramamritham, S. H. Son, and L. C. Dipippo, "Real-time databases and data services," *Real-Time Systems*, vol. 28, no. 2-3, pp. 179–215, 2004.
- [12] C. Lu, X. Wang, and X. Koutsoukos, "Feedback utilization control in distributed real-time systems with end-to-end tasks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 16, no. 6, pp. 550–561, 2005.
- [13] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. Wiley IEEE press, 2004.
- [14] "Oracle Berkeley DB, <http://www.oracle.com/>," 2008.
- [15] "Fire Growth and Smoke Transport Modeling with CFAST, <http://fast.nist.gov/>," 2008.
- [16] X. Jiang, N. Y. Chen, J. I. Hong, K. Wang, L. Takayama, and J. A. L., "Siren: Context-aware computing for firefighting," in *In Proceedings of Second International Conference on Pervasive Computing*, 2004.
- [17] P. Bonnet, J. Gehrke, and P. Seshadri, "Towards sensor database systems," in *MDM '01: Proceedings of the Second International Conference on Mobile Data Management*, 2001.
- [18] S. R. Madden, M. J. Franklin, J. M. Hellerstein, and W. Hong, "Tinydb: an acquisitional query processing system for sensor networks," *ACM Trans. Database Syst.*, vol. 30, no. 1, pp. 122–173, 2005.
- [19] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva, "Directed diffusion for wireless sensor networking," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 2–16, 2003.
- [20] S. Ratnasamy, B. Karp, L. Yin, F. Yu, D. Estrin, R. Govindan, and S. Shenker, "Ght: a geographic hash table for data-centric storage," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 2002.
- [21] G.-S. Ahn, S. G. Hong, E. Miluzzo, A. T. Campbell, and F. Cuomo, "Funneling-mac: a localized, sink-oriented mac for boosting fidelity in sensor networks," in *SenSys '06: Proceedings of the 4th international conference on Embedded networked sensor systems*, 2006.
- [22] P. Desnoyers, D. Ganesan, and P. Shenoy, "Tsar: A two tier sensor storage architecture using interval skip graphs," in *SenSys '05: Proceedings of the 3rd international conference on Embedded networked sensor systems*, 2005.
- [23] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong, "Model-driven data acquisition in sensor networks," in *Proceedings of the 30th VLDB conference*, Toronto, Canada, 2004.
- [24] M. Li, D. Ganesan, and P. Shenoy, "Presto: feedback-driven data management in sensor networks," in *NSDI '06: Proceedings of the 3rd conference on Networked Systems Design & Implementation*, 2006.