Automatic Tuning of Read-Time Tolerances for Optimized On-Demand Data-Streaming from Sensor Nodes

Julius Hülsmann
Technische Universität Berlin
j.huelsmann@tu-berlin.de

Chiao-Yun Li
Fraunhofer FIT
chiao-yun.li@fit.fraunhofer.de

Jonas Traub
Technische Universität Berlin
jonas.traub@tu-berlin.de

Volker Markl
Technische Universität Berlin
volker.markl@tu-berlin.de

ABSTRACT

The Internet of Things provides applications with data streams from billions of sensor devices in real-time. Usually, sensor devices serve multiple queries simultaneously despite having limited computational capabilities. This paper presents a solution for reducing the number of data reads and transmissions by increasing the potential for sharing reads among concurrent streaming queries. Existing read-scheduling techniques on sensor nodes dynamically adjust the data-acquisition rate depending on the data's variability. However, they leave the definition of read-time tolerances to the user. Such read-time tolerances are crucial for sharing reads among queries. We extend previous work by presenting a generally-applicable algorithm that defines read-time tolerances and adapts them on-the-fly depending on observed data characteristics. We evaluate our solution on real-world data and show that it reduces the sensing error by up to 60% compared to existing approaches with the same number of data reads. Respectively, our technique reduces the number of data reads to achieve the same sensing error as existing techniques. To the best of our knowledge, we are the first to automatically set and tune read-time tolerances to reduce sensor readings and data transmissions on sensor nodes.

1 INTRODUCTION

With the advance in communication and information technologies, we can easily access the internet with laptops, tablets, smartphones, and other mobile devices. The network of smart devices, vehicles, and other network-attached sensors forms the Internet of Things (IoT) [16]. In a general IoT setup, multiple sensors are attached to a single device, the sensor node. The sensor node gathers data from the attached sensors and then transmits them to the Stream Processing Engine, enabling it to answer queries.

IoT applications make decisions in real-time based on dynamically changing surroundings. With a vast amount of sensors and the need for real-time data processing, several efficient data acquisition and transmission techniques are proposed to provide data streams to applications [8, 11, 21, 22, 26].

There exist several solutions for making data acquisition and transmission from sensor nodes more efficient. These solutions aim at reducing the number of sensor reads conducted (adaptive sampling) or values transmitted (adaptive filtering) [8]. The key idea of *adaptive sampling* is to modify the read-time frequency based on the recent history of sensor readings. If the values read from the sensor behave unexpectedly (i.e., have a high variability), samples are collected at a higher rate. In contrast, if values barely change, the sampling rate is reduced. *Adaptive filtering* techniques reduce the number of transmitted values by discarding values that evolve predictably. Adaptive sampling and adaptive filtering usually operate on a per-query basis. Instead, *multi-query read-sharing* saves transmissions by scheduling reads that satisfy multiple concurrent queries simultaneously.

Adaptive sampling and adaptive filtering techniques have to be selected and configured in accordance with the corresponding query's *data demand*, i.e., reflecting the consumer's sensitivity to observing changes in the data. It is crucial to note that queries may possess different data demands. For example, the adaptive sampling technique AdaM [23] reacts very fast to abrupt value changes, while FAST [6] incorporates differential privacy features. This is where multi-query optimization becomes necessary.

In previous works, we propose considering the queries' data demand during multi-query optimization by combining adaptive sampling and filtering techniques with multi-query read sharing in a unified framework [9, 20]. User-defined stateful functions (e.g., adaptive sampling techniques) iteratively suggest a query's read-times, and a multi-query read-scheduler exploits read-time tolerances around suggested read-times for sharing reads among queries respecting their data demand.

Figure 1 shows an overview of these two steps of read-time suggestion and read-fusion. As we can see from steps ② and ③, specifying read-time tolerances is essential to read-fusion. However, it is hard for users to manually specify and tune such tolerances. While well-known adaptive sampling techniques provide the desired read-time, they do not define tolerances. Ideally, tolerances for each read request should adjust automatically based on current data characteristics. Intuitively, tolerances should increase if values remain constant or follow an expected trend. Tolerances should shrink if sensor values change rapidly.

In this paper, we introduce an Adaptive Read-Time Tolerance Controller (ARTC), a general algorithm that adjusts read-time tolerances on-the-fly, based on the recent history of sensor readings. ARTC enhances arbitrary adaptive sampling techniques with automated control of read-time tolerances. Such read-time tolerances enable the sharing of sensor values among multiple queries on the same sensor node and thus lead to reading and transmission savings in distributed sensor networks. We design our solution to be generally applicable – independent of the algorithm defining the desired read-times. Therefore, we divide the task of setting read-time tolerances into two parts: Iteratively adapting the read-time tolerance's diameter, and shifting the read-time tolerance interval around the desired read-time: We capture statistics on the data's variability based on which we adapt the read-time tolerance. We use a proportional-integral-derivative controller (PID controller) [2, 3], which is a commonly used form of feedback control [4] to shift the read-time tolerance.

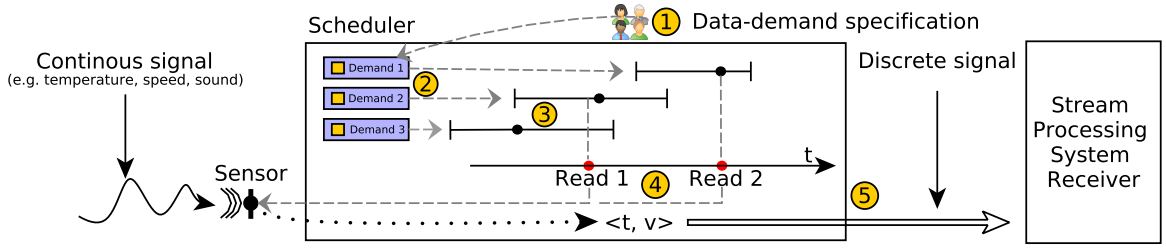


Figure 1: Overview of Optimized On-Demand Data-Streaming from Sensor Nodes [20]: ① The sensor node's scheduler receives the *data demand* of new queries. ② A query's *data demand* defines the next (desired) read-time • at which it requires the next sensor reading and optionally, a read-time tolerance. ③ The scheduler determines the time for the next sensor reading •, fusing requests, if possible. The device wakes up to ④ read a value from the sensor, and ⑤ transmits it to the receivers.

Our implementation is available as an open-source project[1]. We evaluate our solution by simulating sensor sharing based on multiple real-world IoT datasets. We analyze the error and the read-time tolerance induced by read-sharing for different configurations of our algorithm, compare it to setups with fixed read-time tolerances, and show that our solution outperforms fixed read-time tolerance specifications.

The remainder of this paper is structured as follows: We first present our solution in Section 2, which we evaluate in Section 3 on three real-world datasets. After discussing Related Work in Section 4 we conclude in Section 5.

2 ADAPTIVE READ-TIME TOLERANCE

This section explains how our adaptive read-time tolerance controller ARTC adapts the read-time tolerance to the sensed data's variability. We first showcase how to use ARTC in Section 2.1 and subsequently present our solution's internals in Section 2.2.

2.1 The User's Perspective

The user configures ARTC with two parameters: Firstly, E, indicating the permissible inaccuracy in the data-representation relative to the data's average magnitude. Secondly, the optional parameter `dMax`, which specifies a maximal read-time tolerance.

It is essential to highlight that the complexity of using ARTC is transparent to users. They can define streaming queries in declarative languages to the Stream Processing Engine (SPE). The user parametrizes ARTC as part of the query as sketched below:

```
SELECT [t, speed, position]    // SENSOR IDENTIFIERS
FROM [bus 1]                   // SENSOR NODE IDENTIFIER
USING [AdaM]          // READ-TIME SUGGESTION ALGORITHM
WITH [ARTC(E=0.1)] ON [speed]   // CONFIGURATION ARTC
```

As outlined in Figure 1, users can submit queries to the SPE, which then forwards the data demand specification and involved sensors to the specified sensor device ①. The device can thus schedule reads according to the specified data-requirements ③ and to dispatch the data stream to the user ⑤. The user receives a data stream $(t_i, \text{speed}_i, \text{position}_i)_{i \in \mathbb{N}}$ until terminating the query.

2.2 Architecture

We first show how our solution integrates within the multi-query read-scheduling framework executed on the sensor device [20] and then present the internals of ARTC. Table 1 summarizes the newly introduced nomenclature.

[1]https://github.com/TU-Berlin-DIMA/ARTC

Internal Architecture. Once the sensor device receives at least one query from the SPE, it schedules reads, as shown in Algorithm 1. For each new query, the sensor device first performs a read in Line 4. The sensor device then reports the value v gathered from the sensor at time t back to the client who issued the query. It then feeds the read-time and value into the read-time suggestion algorithm (Line 7), which generates the next desired read-time t_D. In the example query in Section 2.1, the read-time suggestion is performed through the adaptive sampling algorithm AdaM. Then, we forward the desired read-time and the last sensor reading to the read-time tolerance algorithm (Line 7), which computes a read-time tolerance by specifying interval boundaries t_s and t_e that enclose the desired read-time. Together with the desired read-time t_D, the interval boundaries constitute the next read request of the query. Lastly, the multi-query read-scheduler schedules reads according to all read requests. We refer the reader to our previous work for details on the optimizations the multi-query read scheduler performs and different optimization objectives [20]. The sensor device reduces its energy consumption by then sleeping until the next read is due. The process of updating the read request is repeated for all queries which received the last sensor reading.

Adaptive Read-Time Tolerance Controller (ARTC). A read-time tolerance algorithm has to meet the following requirements: (i) Despite advances in hardware technologies, sensor devices are still restricted in computational capabilities compared to sinks. As they need to schedule reads very precisely and are often battery-powered, the algorithm has to be energy-efficient. (ii) The algorithm has to adapt to changes in the distribution of the data quickly. Otherwise, the user misses important events. (iii) The algorithm has to be easy to configure and should not be entirely dependent on the read-time suggestion algorithm.

As a lightweight means of keeping track of the distribution of gathered sensor values, we use the iterative Probabilistic Exponentially Weighted Moving Average (PEWMA) algorithm. We use two instances of PEWMA; firstly, we monitor the distribution of the magnitude of sensor readings $\|v\|_2$. We refer to the probabilistic moving average by μ_v and to the estimated standard deviation of the used normal distribution by σ_v. We use the second instance of PEWMA to monitor moving average μ_δ and standard deviation σ_δ of the difference between consecutive read-times $\delta := \|v - v_l\|_2$. In order to be less dependent on the read-time suggestion algorithm, we divide our solution into two parts: (1) the adaptation of the read-time tolerance Δ based on the predictability of the data distribution and (2) shifting of the read-time interval $s \in [-0.5, 0.5]$ based on the value deviation between consecutive sensor readings as shown in Equation 1:

Algorithm 1: Multi-query read-sharing algorithm.

```
t = now();  involvedQueries = newQueries;  u = {};
while running do
    sleepUntil(t);
    v = sensor.read();
    involvedQueries.forwardToSink(t, v);
    while query in involvedQueries do
        t_D = readTimeSuggestion.next(t, v);
        t_s, t_e = readTimeToleranceAlgorithm.next(t, v, t_D);
        u[query] = (t_s, t_D, t_e);
    end
    t, involvedQueries = mqrs.select(u);
end
```

$$t_s = t_D + \Delta \cdot \left(s - \frac{1}{2}\right), \qquad t_e = t_D + \Delta \cdot \left(s + \frac{1}{2}\right) \tag{1}$$

In the next sections, we discuss separately how read-time tolerances Δ and shifts s are computed.

Adaptation of read-time tolerance. We use the variation of δ to assess the predictability of the distribution. If the variation exceeds the threshold E specified by the user,

$$\begin{aligned} & \sigma_\delta && > E \cdot \mu_v \\ \Leftrightarrow \quad & \sigma_\delta - E \cdot \mu_v && > 0, \end{aligned} \tag{2}$$

we reduce the read-time tolerance Δ by the damping factor in Equation 3. This way, it decreases exponentially in the number of consecutive times that the variation exceeds E.

$$\Delta \leftarrow \Delta \cdot \frac{1}{2}. \tag{3}$$

Otherwise, we increase the read-time tolerance by a value proportional to the difference between the user-indicated deviation E and σ_δ, which we denote the step-size δ_E. In order to make the step-size independent of the magnitude of both E and the sensor values, we scale the threshold by the multiplicative inverse of E and μ_v and define the step size in Equation 4.

$$\delta_E := \frac{\sigma_\delta - E \cdot \mu_v}{E \cdot \mu_v} = \frac{\sigma_\delta}{E \cdot \mu_v} - 1. \tag{4}$$

Shifting of read-time interval. We shift the read-time interval around the desired read-time t_D to reduce the algorithm's dependence on the read-time suggestion algorithm's performance. We use a PID controller with `setpoint` E to assess whether the deviation between consecutive sensor readings is within the range specified by the user. That way, we obtain a long-term estimation of the read-time suggestion algorithm's performance and exert limited control over the read-time difference. We scale the controller's output with a factor of 0.1 we determined empirically, which achieves good results for all evaluated datasets.

Complete Algorithm. The overall algorithm is summarized in Algorithm 2: We restate Condition 2 in terms of δ_E and also ensure the user-defined boundaries of Δ and the range of s.

3 EXPERIMENTAL EVALUATION

In this section, we evaluate ARTC on three real-world IoT datasets. We picked the different datasets to portray various data characteristics, which we lay out in Section 3.1. We then introduce our experimental setup in Section 3.2, present the experiments and their results in Section 3.3, and close with a discussion in Section 3.4.

Variable	Description
t,t_l,v,v_l	Current / last sensor reading (time and value).
t_s,t_e	Read-time tolerance interval boundaries.
$t_D \in [t_s, t_e]$	Desired read-time.
δ	eukl. distance between consecutive values.
μ_v,σ_v μ_δ,σ_δ	PEWMA and estimated standard deviation of the magnitude of v, respectively δ.

Table 1: Overview of subsequently used nomenclature.

PEWMA	AdaM	PID Controller	ARTC
$\alpha = 0.5,\ \beta = 0.5,$ $d_{\text{init}} = 20$	$\gamma = 0.2$	$P = 2, I = 2e\text{-}3,$ $D = 0.3$	dMax = 175

Table 2: Configuration of AdaM's and ARTC's components.

3.1 Datasets

We evaluate our solution on three IoT sensor datasets [17]. We provide a brief description for each dataset and state which of the multiple sensors in the dataset we use for our experiments.

The **football monitoring dataset [14]** provided within the scope of the ACM DEBS 2013 Grand Challenge consists of data gathered during a football training game at the Nuremberg Stadium in Germany. We replay the football's absolute velocity in m/s, which is available at 2000 Hz for the match's first half-hour. We evaluate our solution on this dataset as it is very volatile, and the speed of the football spikes abruptly when kicked by a player.

The **daily and sports activities dataset [1]** contains multiple time-series, constituting different individuals performing a total of 19 activities such as sitting, standing, and jumping. We replay data from the torso acceleration sensor in m/s^2, which is available at 25 Hz frequency for 5 minutes per person and activity. We select two daily activities sitting and standing, and two sports activities descending stairs and exercising on a stepper, and concatenate the corresponding time-series. As different activities alternate in the resulting time series, we can observe a drastic shift in the distribution of values each time the performed activity changes.

The **gas dataset [7]** holds concentration levels of dynamic gas mixtures. The data is collected continuously at a frequency of 100 Hz, and we use the first hour of the available data. We replay the gas mixture of ethylene and CO, and the unit of the measurement is *parts per million* (ppm). The variability in this dataset is low, as the distribution of the different chemicals only changes gradually.

3.2 Evaluation Setup

In a production setup, a read-sharing algorithm's performance on a sensor device is measured by counting the number of saved reads of the corresponding query under heavy load. However, this quantity largely depends on the configuration and number of concurrent queries. To assess the performance of read-sharing more objectively, we measure the suggested read-time tolerance Δ instead, while operating only a single query. We simulate heavy load by sampling at random within the proposed read-time tolerance to obtain a realistic view of the induced error. For each experiment, we conduct multiple runs and combine the results.

We record two performance measures during the conducted experiments to evaluate the functionality and effectiveness of ARTC. Firstly, we keep track of the average read-time tolerance μ_Δ, which indicates the read-sharing potential enabled by the algorithm for a total of M suggested read-time tolerances Δ_i: Secondly, we record the induced error for each point in time t, i.e., the distance between the last performed sensor reading $\hat{v}$ prior to t and the actual sensor

Algorithm 2: ARTC.next

Parameters: setpoint E, maximal interval diameter $dMax$
State: shift $s=0$, diameter $\Delta=0$, last value v_l
Input: read-time t, value v, desired read-time t_D
Output: next interval borders $[t_s,t_e]$,

```
μ_v = pewma.next(‖v‖_2);
if v_l is set then
    μ_δ, σ_δ = pewmaDiff.next(‖v−v_l‖_2);
    s += pid.next(μ_δ/μ_v)·0.1;
    δ_E = σ_δ/(μ_v·E) − 1;
    if (δ_E > 0) Δ·= 1/2 else Δ−=δ_E;
    ensure s ∈ [−1/2, 1/2] and Δ ∈ [0,dMax];
else
    Δ=0;
end
t_e,t_s = t_D+Δ·(s± 1/2);
v_l = v;
return t_s,t_e;
```

value v_i at time t. We label ϵ the average error over the entire experiment trace, and denote the error's moving average with window size k in percent of the data's average magnitude via ϵ^k.

During all conducted experiments, we compare the induced error ϵ of ARTC to the error of fixed read-time tolerance algorithms that we configured to achieve the same average read-time tolerance μ_Δ as ARTC. We use the read-time suggestion algorithm AdaM throughout the experiments, and only vary the parameter E of ARTC and the fixed read-time tolerance algorithm accordingly. The parameters that are fixed throughout the experiments are provided in Table 2.

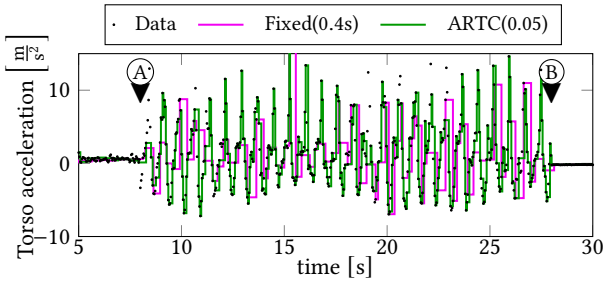
3.3 Experimental Evaluation

We first present the evaluation on an experiment trace on an excerpt from the activities dataset and then provide quantitative results for all three datasets.

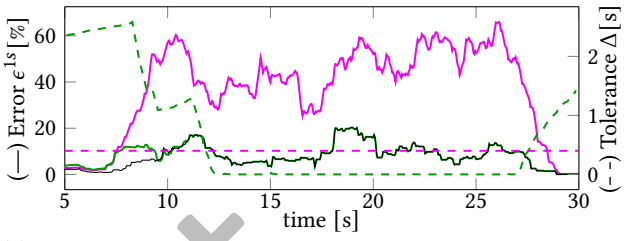
Experiment Trace. Figure 2 shows the performance of ARTC through an experiment trace excerpt from the activities dataset. We execute a total of three query-configurations on the dataset (i) A baseline query without read-sharing, which executes only the read-time suggestion algorithm AdaM, (ii) ARTC configured with $E = 0.05$, and (iii) A fixed read-time tolerance algorithm configured with $\Delta=0.4$ seconds (s).

In Figure 2a, we visualize the raw sensor data at the highest possible rate as black dots. The representation of the sensor provided through configurations (ii) and (iii) is visualized as solid lines, which indicate the last performed sensor reading $\hat{v}_i$ at any given point in time. At any point in time, the difference between the solid lines and the data points is the error induced by the respective query. Figure 2b visualizes the moving average of the relative error over one second and the read-time tolerance Δ_i.

During the visualized experiment trace, a person performs three operations: standing in an elevator still until (A), exercising on a stepper until (B), and sitting. Every time the performed activity changes, we observe a dramatic change in the distribution of the replayed data. While exercising, the variability of the replayed torso acceleration measurements increases significantly.
At the beginning of the experiment, the variability of the data is relatively low. Both the fixed read-time tolerance algorithm and



(a) The black dots show acceleration measurements at the highest available rate. Solid lines visualize the last sensor reading performed with ARTC resp. the fixed read-time tolerance algorithm.



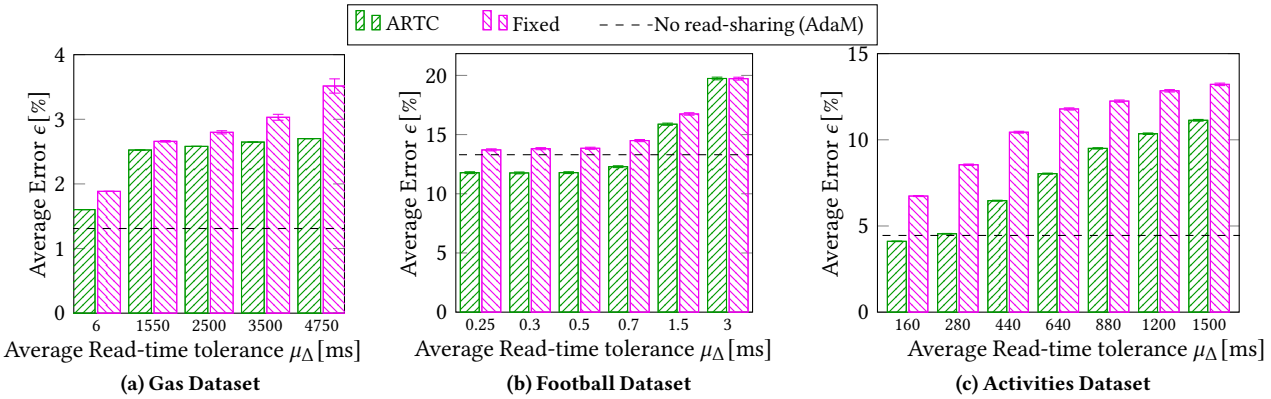
(b) Diameter Δ and 1-second moving average of the error ϵ induced by ARTC, the fixed read-time tolerance algorithm, and without read-sharing (black) altogether. During volatile phases, ARTC decreases Δ and thus reduces ϵ to the error without read-sharing.

Figure 2: Experiment trace of ARTC and baselines.

ARTC achieve a similar error ϵ of less than 5%. However, during this low variability phase, ARTC enables a read-time tolerance larger than 2.4s, which is more than six times larger than those achieved by the fixed read-time tolerance algorithm.
A brief spike in the torso acceleration precedes the individual's exercising routine at the 5-second mark, which leads to ARTC reducing the read-time tolerance. Once the torso acceleration starts to oscillate ((A)), ARTC reduces the read-time tolerance to a minimum, such that the error levels with those induced by the read-time tolerance algorithm AdaM. During this phase, the read-time tolerance proposed by the fixed algorithm is higher than those proposed by ARTC at the cost of an error ϵ of approximately 50% of the data's magnitude, while ARTC causes no additional error. Once the individual sits ((B)), ARTC increases the read-time tolerance again.

By controlling the read-time tolerance based on the data's variability, ARTC is able to outperform the fixed read-time tolerance algorithm in both depicted performance metrics: When considering the entirety of the depicted trace, the read-time tolerance of the fixed read-time tolerance algorithm is approximately doubled by ARTC, while ARTC can reduce the average error by 75%.

Quantitative Experiments. To assess the performance of ARTC under different circumstances, we run multiple experiments on the gas, activities, and football datasets. We compare ARTC to fixed read-time tolerance algorithms achieving the same average read-time tolerance Δ in Figure 3. Throughout the experiments, we observe that ARTC outperforms or at least levels the fixed read-time tolerance algorithm. Overall, ARTC outperforms the fixed read-time tolerance algorithm by the largest margin on the activities dataset: on average, it reduces the error by 31% compared to approximately 15% on the other two datasets. However, the additional error induced by ARTC for enabling read-sharing is smallest on the gas dataset (1.3 percentage points for $\Delta > 4$s). This relatively small induced error reflects the relatively steady nature



(a) Gas Dataset **(b) Football Dataset** **(c) Activities Dataset**

Figure 3: Performance of queries with different configurations of ARTC and the two baselines on the introduced datasets.

of the gas data. We discuss the experimental results in detail for each dataset in the following paragraphs.

The gas dataset (Figure 3a) contains sensor values with the lowest variability of all three datasets. The error of the read-time suggestion algorithm AdaM without read-sharing is approximately 1.4% of the data's average magnitude. With the configurations of ARTC visualized in Figure 3, large read-time tolerances of up to 4.75s are possible.
While small deviations on the dataset can only be captured when enabling read-sharing only to a limited extent of 6ms, we observe that ARTC is able to allow for large read-time tolerances of 2−5s without a significant increase in the average error. On the other hand, the fixed read-time tolerance algorithm's performance deteriorates, such that the error of the fixed algorithm is 30% larger than those caused by ARTC for a read-time tolerance of 4.75s. The additional error of read-sharing through ARTC stays below the error induced by the read-time suggestion algorithm for all executed configurations.

The football dataset (Figure 3b) has the highest variability of all presented datasets. The error of the read-time suggestion algorithm alone is 13% of the data's average magnitude. As most of the configurations of ARTC indicate an average error below or at the same level as the error of the read-time suggestion algorithm, read-sharing is only enabled to a very limited extent, up to 3ms. On this dataset, we observe that ARTC is able to reduce the error of the read-time tolerance algorithm slightly by up to 13% when enabling read-sharing to a limited extent. For read-time tolerances of 1.5ms, the error of the query configured with ARTC (15.8%) draws nearer to those of the fixed algorithm (16.7%). For a read-time tolerance of 3ms, the performance of both algorithms levels.

The read-time suggestion algorithm on the activities dataset (Figure 3c) has an error of 4.4%, matched for ARTC up to a read-time tolerance of 280ms. The executed configurations of ARTC achieve a read-time tolerance of up to 1.5s. On this dataset, ARTC outperforms the fixed read-time suggestion algorithm by the largest margin. ARTC is able to adapt to changes in the variability of the data during the experiment that occur whenever the individuum starts performing a different activity. While the error of the fixed read-time tolerance algorithm is approximately 90% larger than those of ARTC for a read-time tolerance of 280ms, the margin between the performance of both algorithms continuously shrinks to 20% for a read-time tolerance of 1.5s. This is because the algorithm misses small fluctuations in the data and takes a long time to adapt to brief periods of activity, which only last for several seconds.

3.4 Discussion

The evaluation of the single trace experiment executed on the activities dataset shows that ARTC can quickly adapt to a change in the distribution of values gathered from the sensor. During the quantitative analysis conducted on the three different datasets, ARTC outperforms the corresponding fixed configuration. For our experiment configurations, it is even possible to enable read sharing to a limited extent without a detrimental effect on the accuracy of the data representation altogether.

4 RELATED WORK

To the best of our knowledge, we are the first to propose a general, adaptive demand-based read-time tolerance controller. However, the underlying problem of reducing the number of reads and transmissions has been studied from various angles. Therefore, we first present existing work on read- and transmission sharing. We then present adaptive sampling techniques, as they are an integral building block of ARTC and share a similar objective.

Read- and transmission sharing. TinyDB [12] introduces the concept of acquisitional query processing (ACQP) to control the sampling frequency during *Single-Query Optimization*. ACQP arranges database operators and sensor readings in a common processing pipeline. Operators with low selectivity reduce the acquisition frequency by filtering out sensor readings before succeeding read operations. However, TinyDB only allows for periodic sampling algorithms at the processing pipeline's source.

Multiple approaches known from the literature conduct *spatial resource sharing*, i.e., optimize the set of deployed queries through a global view into a single query. Li et al. [10] allow the user to define query priorities and -deadlines. They fuse aggregate queries accordingly, perform utility-driven compression, and global transmission-scheduling to save reads and transmissions. However, they do not adapt to the variability of the distribution in scheduling and fusing sensor readings.

As opposed to spatial resource sharing, *data sharing techniques* perform local optimizations using a single sensor reading for multiple queries [13, 24, 25]. Tavakoli et al. [18] model the overlaps of tolerance intervals in an online evolving interval-cover graph, which they use to determine read-times. All four approaches are unable to adapt to changes in the distribution of the data both in scheduling reads and in computing read-time tolerances. We advanced data sharing in the context of adaptive sampling techniques [19, 20]. To that end, we proposed to combine demand-based adaptive read-time suggestion and read-time fusion in a

single framework. We were able to report savings of 87% in reads and transmissions, which we further increase through ARTC.

Adaptive sampling tailors the sampling frequency to the distribution of the data [8]. Padhy et al. propose a confidence-based adaptive sampling method called Utility-based Sensing and Communication (USAC) [15]. They apply linear regression to predict the next sensor value with a bounded error-range, the so-called confidence interval (CI). If a value is outside the CI, the sensor starts sampling at maximal frequency. Otherwise, the frequency decreases exponentially by a factor $\alpha \in [0,1]$ until it reaches the minimum sampling frequency.

Aiming to provide an energy-efficient solution in the realm of Big Data and IoT, Trihinas et al. propose the Adaptive Monitoring Framework (AdaM) [23]. They use an ad-hoc forecasting method called PEWMA to produce one-step forecasts, which they then use to compute the metric stream's variability.

Fan et al. propose Filtering and Adaptive sampling for Differentially Private Time Series Monitoring (FAST) [6]. They define a so-called privacy budget to add Laplace noise to the original observations to achieve differential privacy. Then they generate estimates, the quality of which is then used by the sampling component to adjust the sampling rate using a PID controller. Compared to AdaM and USAC, FAST is slower to adapt to changes in the distribution of the data but achieves comparable results.

We design our adaptive read-time tolerance controller ARTC using ideas from all three of the aforementioned adaptive sampling algorithms. We use a PID controller [3] in order to assess whether the read-time suggestion algorithm achieves the data-quality ARTC targets. Similar to AdaM, we use PEWMA [5] in order to monitor the distribution of samples. We use the idea presented by Padhy et al. of decreasing tolerances rapidly if the desired data accuracy is missed, which enables us to adapt to changes in the distribution quickly.

5 CONCLUSION

We previously developed a multi-query read-scheduling algorithm that enables adaptive sampling in a sensor network [20]. In this paper, we now extend our work by proposing the easy to configure adaptive read-time tolerance controller ARTC. Our experimental evaluation shows that ARTC tailors the extent of read-sharing to the data-accuracy demands of end-applications. ARTC is generally-applicable for defining read-time tolerances when scheduling sensor read-times. Thus, it enables multi-query optimization through sharing sensor readings for arbitrary adaptive sampling techniques. We evaluate ARTC on three real-world IoT datasets with different data characteristics and shifts in the data distribution. Our solution reduces the error in the representation by up to 60% compared to fixed read-time tolerance algorithms by adapting to the sensed data's variability. ARTC not only reduces the number of reads and transmissions while achieving the same sensing error as existing techniques, but also enables multi-query read-sharing for queries issued by users without domain-knowledge. We make our code and evaluation available open-source. We also provide detailed instructions on how to execute custom experiments. In our previous work, we allow the user to define a so-called penalty-function [20] to further increase the read-sharing potential under specific circumstances. We plan to extend our solution to adaptively tune such penalty-functions based on the data's variability as well.

Acknowledgements: This work has been supported by German Ministry for Education and Research as BIFOLD (01IS18025A, 01IS18037A).

REFERENCES

[1] Kerem Altun and Billur Barshan. 2010. Human activity recognition using inertial/magnetic sensor units. In *International workshop on human behavior understanding*. Springer, 38–51.

[2] M Araki. 2009. PID control. *Control Systems, Robotics and Automation: System Analysis and Control: Classical Approaches II, Unbehauen, H.(Ed.). EOLSS Publishers Co. Ltd., Oxford, UK., ISBN-13: 9781848265912* (2009), 58–79.

[3] Karl J Åström and Tore Hägglund. 2006. Advanced PID control. In *The Instrumentation, Systems, and Automation Society*. Citeseer.

[4] A Zul Azfar and Desa Hazry. 2011. A simple approach on implementing imu sensor fusion in pid controller for stabilizing quadrotor flight control. In *7th International Colloquium on Signal Processing and its Applications*. IEEE, 28–32.

[5] Kevin M Carter and William W Streilein. 2012. Probabilistic reasoning for streaming anomaly detection. In *Statistical Signal Processing Workshop (SSP)*. IEEE, 377–380.

[6] Liyue Fan, Li Xiong, and Vaidy Sunderam. 2013. FAST: differentially private real-time aggregate monitor with filtering and adaptive sampling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 1065–1068.

[7] Jordi Fonollosa, Sadique Sheik, Ramon Huerta, and Santiago Marco. 2015. Reservoir computing compensates slow response of chemosensor arrays exposed to fast varying gas concentrations in continuous monitoring. *Sensors and Actuators B: Chemical* 215 (2015), 618–629.

[8] Dimitrios Giouroukis, Alexander Dadiani, Jonas Traub, Steffen Zeuch, and Volker Markl. 2020. A Survey of Adaptive Sampling and Filtering Algorithms for the Internet of Things. In *DEBS'20: 14th ACM International Conference on Distributed and Event-Based Systems*.

[9] Julius Hülsmann, Jonas Traub, and Volker Markl. 2020. Demand-based Sensor Data Gathering with Multi-Query Optimization. In *Proceedings of the 46th Conference on Very Large Databases*, Vol. 13. VLDB Endowment. Issue 12.

[10] Ming Li, Tingxin Yan, Deepak Ganesan, Eric Lyons, Prashant Shenoy, Arun Venkataramani, and Michael Zink. 2007. Multi-user data sharing in radar sensor networks. In *Proceedings of the 5th international conference on Embedded networked sensor systems*. ACM, 247–260.

[11] Chong Liu, Kui Wu, and Jian Pei. 2007. An energy-efficient data collection framework for wireless sensor networks by exploiting spatiotemporal correlation. *IEEE transactions on parallel and distributed systems* 18, 7 (2007), 1010–1023.

[12] Samuel R Madden, Michael J Franklin, Joseph M Hellerstein, and Wei Hong. 2005. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on database systems (TODS)* 30, 1 (2005), 122–173.

[13] Rene Muller and Gustavo Alonso. 2006. Efficient sharing of sensor networks. In *International Conference on Mobile Ad Hoc and Sensor Systems*. IEEE, 109–118.

[14] Christopher Mutschler, Holger Ziekow, and Zbigniew Jerzak. 2013. The DEBS 2013 grand challenge. In *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM, 289–294.

[15] Paritosh Padhy, Rajdeep K Dash, Kirk Martinez, and Nicholas R Jennings. 2006. A utility-based sensing and communication model for a glacial sensor network. In *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. ACM, 1353–1360.

[16] Chana R Schoenberger. 2002. The internet of things. *Forbes* (2002), 155160–155160.

[17] Amit Kumar Sikder, Giuseppe Petracca, Hidayet Aksu, Trent Jaeger, and A Selcuk Uluagac. 2018. A Survey on Sensor-based Threats to Internet-of-Things (IoT) Devices and Applications. *arXiv preprint arXiv:1802.02041* (2018).

[18] Arsalan Tavakoli, Aman Kansal, and Suman Nath. 2010. On-line sensing task optimization for shared sensors. In *Proceedings of the 9th ACM/IEEE International Conference on Information Processing in Sensor Networks*. ACM, 47–57.

[19] Jonas Traub. 2019. *Demand-based data stream gathering, processing, and transmission*. Ph.D. Dissertation. Technische Universität Berlin.

[20] Jonas Traub, Sebastian Breß, Tilmann Rabl, Asterios Katsifodimos, and Volker Markl. 2017. Optimized On-demand Data Streaming from Sensor Nodes. In *Proceedings of the 2017 Symposium on Cloud Computing*. ACM, 586–597. https://doi.org/10.1145/3127479.3131621

[21] Jonas Traub, Julius Hülsmann, Sebastian Breß, Tilmann Rabl, and Volker Markl. 2019. SENSE: Scalable Data Acquisition from Distributed Sensors with Guaranteed Time Coherence. *arXiv preprint arXiv:1912.04648* (2019).

[22] Demetris Trihinas, Luis F Chiroque, George Pallis, Antonio Fernández Anta, and Marios D Dikaiakos. 2018. ATMoN: Adapting the" Temporality" in Large-Scale Dynamic Networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems*. IEEE, 400–410.

[23] Demetris Trihinas, George Pallis, and Marios D Dikaiakos. 2015. AdaM: An adaptive monitoring framework for sampling and filtering on IoT devices. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 717–726.

[24] Shili Xiang, Hock Beng Lim, and Kian-Lee Tan. 2006. Impact of multi-query optimization in sensor networks. In *Proceedings of the 3rd workshop on Data management for sensor networks: in conjunction with VLDB 2006*. ACM, 7–12.

[25] Shili Xiang, Hock Beng Lim, Kian-Lee Tan, and Yongluan Zhou. 2007. Two-tier multiple query optimization for sensor networks. In *27th International Conference on Distributed Computing Systems (ICDCS'07)*. IEEE, 39.

[26] Steffen Zeuch, Ankit Chaudhary, Bonaventura Del Monte, Haralampos Gavriilidis, Dimitrios Giouroukis, Philipp M Grulich, Sebastian Breß, Jonas Traub, and Volker Markl. 2020. The NebulaStream Platform: Data and application management for the internet of things. *CIDR* (2020).