Data Quality and Query Cost in Pervasive Sensing Systems

David Yates  
Computer Information Systems Dept.  
Bentley College  
Waltham, Massachusetts 02452  
dyates@bentley.edu

Erich Nahum  
IBM T.J. Watson Research Center  
19 Skyline Drive  
Hawthorne, New York 10532  
nahum@watson.ibm.com

Jim Kurose and Prashant Shenoy  
Dept. of Computer Science  
University of Massachusetts  
Amherst, Massachusetts 01003

Abstract

This research is motivated by large-scale pervasive sensing applications. We examine the benefits and costs of caching data for such applications. We propose and evaluate several approaches to querying for, and then caching data in a sensor field data server. We show that for some application requirements (i.e., when delay drives data quality), policies that emulate cache hits by computing and returning approximate values for sensor data yield a simultaneous quality improvement and cost savings. This win-win is because when system delay is sufficiently important, the benefit to both query cost and data quality achieved by using approximate values outweighs the negative impact on quality due to the approximation. In contrast, when data accuracy drives quality, a linear trade-off between query cost and data quality emerges. We also identify caching and lookup policies for which the sensor field query rate is bounded when servicing an arbitrary workload of user queries. This upper bound is achieved by having multiple user queries share the cost of a single sensor field query. Finally, we demonstrate that our results are robust to the manner in which the environment being monitored changes using models for two different sensing systems.

1 Introduction

Applications for pervasive sensing systems vary in scale from monitoring and controlling microscopic manufacturing equipment, to implementing an earthquake early warning system for a country like Japan. There are many performance metrics of interest in sensing systems for such applications. We focus on two that are common to the vast majority of sensing applications:

1. The accuracy of the data acquired by the application from the sensor networks; and

2. the total system end-to-end delay incurred in the sequence of operations needed for an application to obtain sensor data.

Although almost all pervasive sensing applications have performance requirements that include accuracy and system delay, their relative importance may differ between applications. We therefore define the quality of the data provided to sensing applications to be a combination of accuracy and delay. As in most systems, improved quality usually comes at some cost. For current wireless sensor networks, the most important component of cost typically is the energy consumed in providing the requested data. In turn this is dominated by the energy required to transport messages through the sensor field. This cost versus quality trade-off has recently been an active area of research [2, 17, 18, 19, 23].

To perform our research, we construct a model for a pervasive sensing system. We then develop novel policies for caching sensor network data values in sensor field gateway servers, and then retrieving these values via cache lookups. We also propose a new objective function for data quality that combines accuracy and delay. Finally, we use our sensing system model to assess the impact of several factors on data quality and query cost performance:

- Our caching and lookup policies; the
- relative importance of data accuracy and system end-to-end delay; and the
- manner in which the sensed data values in the environment change.
This assessment evaluates seven different caching and lookup policies by implementing them in a simulator based on CSIM 19 [15, 16].

Almost all sensing system deployments have three main components:

1. One or more sensor fields consisting of sensor field nodes that communicate with one or more base stations;
2. One or more data servers (or gateways) that accept requests for sensor data and generate replies for these requests; and
3. Monitoring and control centers that are connected to the appropriate sensor data servers via a backbone network.

Figure 1 shows an example of such a deployment with two sensor fields, one data server, and one monitoring and control center. If the data server shown in this figure is augmented with storage, it can store and cache sensor field values that are carried in query replies.

The caching approaches we propose are designed to be general since they make no assumptions about whether the sensor network architecture uses a structured or unstructured data model. In other words, our approaches are independent of the database model for the sensor network. The database could implement a structured schema that extends a standard like the Structured Query Language (SQL). The TinyDB and Cougar systems both advocate this approach [4, 12]. However, the schema could also be modified while the system is running (e.g., as in IrisNet [6]). The database model might also expose a low-level interface to the application. Directed Diffusion [8] does this by allowing applications to process attributes or attribute-value pairs directly.

1.1 Data Acquisition and Caching in Pervasive Sensing Systems

Consider the impact of adding a cache to the data server or gateway in Figure 1. Figure 2 shows such a system in which a cache is added to the internal architecture of the server, on the “border” between the sensor field(s) and the backbone network. There are two possible data paths that can be traversed in response to a query from the backbone network:

- For a cache miss, a query is sent to the sensor field by the gateway, incurring a cost. To update the cache, each sensor data value \( v_i \) is copied into a cache entry. A cache entry, \( e_l \), associates with location \( l_i \), the most recent value observed at this location, and its timestamp into the tuple \( \langle l_i, v_i, t_i \rangle \). We say that the system delay, \( S_d \), is the time between an application query arriving at the point labeled \( \text{Query}_m \) in Figure 2 and the corresponding reply departing from \( \text{Reply}_m \). The value deviation, \( D_v \), is the unsigned difference between the data value in \( \text{Reply}_m \) and the true value at \( l_i \) when \( \text{Reply}_m \) leaves the gateway reply queue.

- The data path for a cache hit is much shorter than for a cache miss. For example, if the cache is indexed by location, and a cache entry is present for a location \( l_i \) specified in a query, a reply can be generated using only the information in the tuple that corresponds to \( l_i \). Since the processing required to perform this cache lookup and generate a reply is relatively small, we assume that the system delay for a cache hit (\( S_d \)) and its associated cost are both zero. We also determine the value deviation for cache hits (\( D_v \)) in the same way as for cache misses.

We exploit spatial locality within sensor field data in the cache. Specifically, some caching and lookup policies allow cache “hits” in which the value at location \( l_i \) is approximated based on values \( v_{l_i'} \) from neighboring location(s) \( \{l_i' \in N(l_i)\} \). (Here \( N(l_i) \) denotes the neighborhood of location \( l_i \).) We develop and describe three such policies that implement what we call approximate lookups and queries. We compare these approximate policies with four precise lookup and query policies that only use information associated with location \( l_i \) to process queries that reference location \( l_i \).
2 Cost and Quality in Sensing Systems

2.1 Caching and Lookup Policies

Our caching and lookup policies are designed to explore alternative techniques for increasing the effective cache hit ratio, and thus conserving sensing system resources.

All of the caching and lookup policies we propose and evaluate incorporate an age threshold parameter $T$ that specifies how long each entry is stored in the cache. We now describe all seven of our caching and lookup policies.

- **All hits** (age threshold parameter $T = \infty$): In this policy cache entries are loaded into the cache but are never deleted, updated, or replaced.

- **All misses** (age parameter $T = 0$): In this policy entries are not stored in the cache.

- **Simple lookups** ($T$): This caching policy results in a cache hit or cache miss based on a lookup at the location specified in each user query. If consecutive misses occur in the cache for the same location, this policy sends redundant queries into the sensor field. When a reply is received its value is loaded into the cache, stored for $T$ seconds, and then deleted.

- **Piggybacked queries** ($T$): A cache hit or miss is determined only by a lookup at the location specified in the user query. If a query has already been issued to fill the cache at a particular location, subsequent queries block in a queue behind the original query and leverage the pending reply to fulfill multiple queries.

- **Greedy age lookups** ($T$): A cache hit or miss is determined by a lookup first at the location specified in the query, and second by lookups at all neighboring locations. If there is more than one neighboring cache entry, the freshest (newest) cache entry is selected. As for piggybacked queries, if a query has already been issued to fill the cache at any of these locations, subsequent queries block in a queue behind the original query and leverage the pending reply to fulfill multiple queries. This is also true for the last two policies: greedy distance lookups and median-of-3 lookups.

- **Greedy distance lookups** ($T$): A cache hit or miss is determined by a lookup first at the location specified in the query, and second by lookups at neighboring locations. If there is more than one neighboring cache entry, the nearest cache entry is selected.

- **Median-of-3 lookups** ($T$). A cache hit or miss is determined by a lookup first at the location specified in the query, and second by lookups at all neighboring locations. If there are at least three neighboring cache entries, the median of three randomly selected entries is selected as the value returned with a cache hit. If there are one or two neighboring cache entries, a randomly selected entry provides a cache hit. Otherwise, the query is treated as a miss.

By implementing blocking behind pending sensor field queries, four of these seven policies have an upper bound on the sensor field query rate, $R_f$. Specifically,

$$\max(R_f) = \frac{|N|}{T}. \quad (1)$$

The four policies are piggybacked queries, median-of-3 lookups, and the two approximate greedy policies. In Equation (1), $|N|$ is the number of distinct greedy policies that can be specified in queries for sensor data.

2.2 Sensing System Data Quality and Query Cost

We normalize sensor network data quality in order to compare quality measurements from different sensing systems, as well as for different system parameters (e.g., number of sensors, distance between sensors, etc.) We define data quality to be a linear combination of normalized system delay and normalized value deviation using a parameter $A$, which is the relative importance of delay when compared with value deviation. The expression that defines quality, denoted $Q_n$, is:

$$Q_n = A \frac{1}{1 + e^{-b}} + (1 - A) \frac{1}{1 + e^{-c}} \quad (2)$$

where $-b$ and $-c$ are the exponents used to perform softmax normalization on delays and value deviations, and $0 \leq A \leq 1$. The exponents in Equation (2) are the $z$ scores of their respective values and are therefore defined as follows:

$$-b = -\frac{S_d - \text{mean}(S_d)}{\text{stddev}(S_d)}, \quad (3)$$

$$-c = -\frac{D_v - \text{mean}(D_v)}{\text{stddev}(D_v)}. \quad (4)$$

Since small values of system delay ($S_d$) and value deviation ($D_v$) are both desirable, smaller values of $Q_n$, e.g., $0 < Q_n \ll 0.5$ imply better data quality, and larger values of $Q_n$ correspond to worse quality. Softmax normalization yields transformed values that lie in the range $[0, 1]$. Because of this property, and because of our definition of $A$, $0 \leq Q_n \leq 1$. This type of normalization has been used by others in neural networks; data mining for pattern recognition; and data classification [1, 3, 7, 14].

We use two different sensing system models in our research in order to generalize our results. The first model uses correlated random variables to simulate how the environment changes for 100 sensor locations. This model
gives us the flexibility to vary how the environment changes. The second model uses real-world trace data to drive how the environment changes. This trace data was taken from 54 light, temperature, and humidity sensors deployed in the Intel Berkeley Research lab over a five-week period [5].

2.3 Simulated Changes to the Environment

For the simulated changes to environment, the sensor field is a 3-dimensional field with rectangular planes on six faces. There is an 8-unit spacing between 10 sensors in the X-dimension, a 6-unit spacing for 10 sensors in the Y-dimension, and a 4-unit spacing for 10 sensors in the Z-dimension. Four base stations are placed on the X-Y plane. These four base stations are then connected to the gateway server that has the common cache. Sensors always communicate with their closest base station at a cost that incorporates free-space energy loss for each transmission [13]. Thus, the properties of each one-way communication to and from location \( l \) are as follows:

\[
Cost_l = pr^2_{b_l} \quad \min(Cost_l) = 1 \text{ unit} \quad (5)
\]

where \( r_{b_l} \) is the distance between location \( l \) and its nearest base station \( b_l \), and \( p \) is the normalization constant for the set of costs. In addition,

\[
Delay_l = qr_{b_l} \quad \max(Delay_l) = 1 \text{ second} \quad (6)
\]

where \( q \) is the normalization constant for the set of delays. We assume that all four base stations communicate with the gateway server containing the cache at zero cost, with zero delay, and using infinite bandwidth. Thus, the minimum cost to query a location in the sensor field is normalized to 2 units (1 for the query + 1 for the reply), and the maximum delay to query a location in the sensor field is 2 seconds (not including queuing delay). Finally, each base station is connected to the sensor field with an access link with a capacity of 25 queries per second.

2.4 Trace-driven Changes to the Environment

For the trace-driven changes to the environment, our second sensor field model has more than an order of magnitude fewer locations (54 instead of 1000). The sensors are arranged in a 2-dimensional field at the numbered locations in Figure 3, which is taken from [5]. Each entry in the trace is from a Mica2Dot sensor, which senses humidity, temperature, light, and battery voltage. The trace contains over 2.2 million entries taken over more than five weeks in early 2004. This means that one location reads and records new sensor field values an average of about once every 1.33 seconds. We wanted to use the most dynamically changing of the sensor field values in our model to maximize the error in query accuracy. We therefore chose the value with the largest average difference between samples. This was light intensity, which is reported in Lux. A value of 1 Lux corresponds to moonlight, 400 Lux to a bright office, and 100,000 Lux to full sunlight.

Four base stations are placed at the corners of the floor plan shown in Figure 3. As before, sensors always communicate with their closest base station. We further assume that the cost and delay of each one-way communication are given by Equations (5) and (6), respectively.

2.5 Query Workload Model

We use a query workload model that is well suited for pervasive sensing applications that include monitoring and control functions. Many of these applications have a workload that includes a periodic arrival process of queries as well as a random arrival process. There are examples of query workloads that capture both of these components in the literature, e.g., [8, 9, 20]. On the other hand, other researchers assume that queries either have exclusively periodic interarrival times [11, 12] or random (usually exponential) interarrival times [4, 24]. We assume that the query workload for our applications consists of the superposition of two query processes: a polling component that slowly scans the sensor field at a fixed rate, and a random component that consists of queries to different locations in the sensor field. Within this random component it is equally likely that each location in the sensor field will be sampled. This workload model is similar to models used by others in [8, 9, 20]. Specifically, our query workload is characterized by two parameters:

- \( \tau \) = the period of the polling component of the query workload (\( \tau > 0 \)); and
- \( \lambda \) = the average query arrival rate of a process that represents the random component of our workload.

For simulated changes to the environment, \( \lambda \) and \( \tau \) are fixed: \( \lambda = 81 \) queries per second is used as the rate pa-
rameter to generate queries with exponentially distributed interarrival times with mean $1/\lambda$. The parameter $\tau$ is set to 111.1T seconds so that the arrival rate for polling queries is 9 queries per second. When $\lambda = 81$ and $\tau = 111.1T$, the aggregate arrival rate for queries is $81 + 9 = 90$ queries per second. Since the total capacity of the sensor field access links is $4 \times 25 = 100$ queries per second, their average link utilization is 0.90 for “all miss” runs, and less for runs that include some cache hits.

For trace-driven changes to the environment, $\lambda$ and $\tau$ are fixed for the results described in Section 3: $\lambda = 0.81$ queries per second and $\tau = 600$ seconds. This makes the average arrival rate for queries two orders of magnitude less than query rate for simulated changes, namely 0.9 queries per second. The average link utilization is also 0.90 for “all miss” runs. These parameters are varied in Sections 4 and 5 as we examine performance trends in our two different sensing system models.

3 Discussion of Results

We wanted our simulated results to capture the fact that sensor field readings are correlated in both space and time. In our sensor field model, at time $t + 1$, the value at each location $l$ is drawn from a normal distribution with mean

$$\mu_{l,t+1} = \frac{1}{3} \mu + \frac{1}{3} \mu_{l,t} + \frac{1}{3} \mu_{N(l),t}.$$  

The long-term mean of this distribution is $\mu = 0$. The standard deviation $\sigma = 0.407514$, and the tails are truncated at minimum / maximum values of $\sigma - 6\mu / \sigma + 6\mu$. This standard deviation is the same as the standard deviation of the system end-to-end delays during a set of 20 runs without a cache for our 1000-node sensor network model. $N(l)$ denotes the neighbors of location $l$, and each neighboring location $l'$ of $l$ contributes to $\mu_{N(l),t}$ in proportion to $\mu_{l,t}/r_{l,l'}$, where $r_{l,l'}$ is the distance between locations $l$ and $l'$. This model for a changing environment is based on the model for correlated sensor network data developed by Jindal and Psounis [10].

Each data value presented in our results is derived by averaging 20 simulation runs initialized with different seeds. Additional details of our experimental methodology are described in [22].

The odd-numbered figures, Figures 5, 7 and 9, show results for light intensity in Lux measured over time in the Intel Berkeley lab data set. These results are for $T = 90$ seconds, and 0.9 queries per second. Note that because of Equation (1), the maximum sensor field query rate, $\max(R_f)$, is reduced to $54/90 = 0.6$ queries per second.

The even-numbered figures, Figures 4, 6 and 8, are for correlated changes to the environment with both the age parameter, $T$, and the average query rate scaled for the more
rapidly changing environment. Specifically, $T = 8.88$ seconds and the average user query rate is 90 queries per second. Because of Equation (1), the maximum sensor field query rate $\max(R_f) = 1000/T = 112.5$ queries per second.

We can draw two main conclusions from our experiments using the correlated and trace-driven models for how the environment changes. Results from these experiments appear in Figures 4 through 11.

1. **There is a cost vs. quality trade-off for some data quality requirements but not others.** For example, consider the results shown in Figures 4 and 5. Figure 5 shows cost versus quality for all seven caching and lookup policies, where $A = 0.1$ and the values at each location are changed according to the lab trace [5]. At the smallest cost, we have a 100% cache hit ratio (labeled “All hits”) that provides a quality of below 0.6 for zero cost. For the largest cost, we see that a 0% cache hit ratio (labeled “All misses”) provides the third-best quality at a cost of approximately 19 units. Recall that for quality, smaller values indicate better quality. The remaining five caching and lookup policies provide a linear trade-off between cost and quality. Figure 4 also shows a trade-off between cost and quality for the same value of $A$ and the same seven caching and lookup policies, but with changes to the environment now modeled by a series of values correlated in space and time. There are two observations worth noting when comparing these first two figures. First, the cost values in Figure 5 are less than in Figure 4 because the distances within the sensor field are smaller. Second, the trends are similar between these two figures, with the exception of the increase in quality of the “all misses” policy between Figure 4 and Figure 5. This worse “all misses” quality is due entirely to an increase in the normalized delay term in the left hand side of Equation (2). This can be verified by comparing the relative differences in delays between the policies, shown on the horizontal axes in Figures 6 and 7.

We now examine system configurations for which delay is the more important component of quality in more detail. Figures 10 and 11 show such configurations for a value of $A = 0.9$. The most remarkable result in these figures is that there is no trade-off between cost and quality when we significantly prioritize delay over value deviation. The two greedy caching and lookup policies have the best cost performance and the best quality performance for both models of changing the environment in Figures 10 and 11. Even though the “all hits” policy has the best absolute performance in these figures, we don’t consider this a practical policy since it never updates the cache.

In studying Figures 4 through 11 it is interesting to understand which system variables depend on which system parameters. For example, cost, delay, and hit ratio values in these simulation results each depend on the following three variables:

- The caching and lookup policies themselves (including the value of $T$); the
- physical configuration of the sensor field; and the
- query arrival process.

Thus, a cost vs. delay or a cost vs. hit ratio graph is the same for different experiments in which these three variables are held constant. To see how cost and delay both increase with lower cache hit ratios, Table 1 shows the cache hit ratio for each of the (cost, delay) points in Figure 6. Similarly, Table 2 shows the hit ratio for each of the (cost, delay) points in Figure 7.

![Figure 10. Cost vs. Quality for $A = 0.9$ and Correlated changes over 1000 locations.](image1)

![Figure 11. Cost vs. Quality for $A = 0.9$ and Trace-driven changes over 54 locations.](image2)

<table>
<thead>
<tr>
<th>Policies</th>
<th>Hit ratio</th>
<th>Cost</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>All hits</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>All misses</td>
<td>0</td>
<td>94</td>
<td>1.18</td>
</tr>
<tr>
<td>Simple lookup</td>
<td>0.40</td>
<td>56</td>
<td>0.69</td>
</tr>
<tr>
<td>Greedy age lookup</td>
<td>0.62</td>
<td>37</td>
<td>0.39</td>
</tr>
<tr>
<td>Greedy distance lookup</td>
<td>0.60</td>
<td>38</td>
<td>0.44</td>
</tr>
<tr>
<td>Median-of-3 lookup</td>
<td>0.55</td>
<td>43</td>
<td>0.51</td>
</tr>
<tr>
<td>Piggyback queries</td>
<td>0.40</td>
<td>57</td>
<td>0.69</td>
</tr>
</tbody>
</table>

Value deviation depends on the same parameters listed above, and additionally on the manner in which the environment changes. Thus, cost vs. value deviation graphs are the same when the policies, sensor field, query arrival process, and method for changing the environment are all identical. Figure 8 and Figure 9 show cost vs. value deviation results for correlated changes and trace-driven changes to the environment, respectively. The most interesting difference between the two figures is the overall increase in the dispersion of the value deviations in Figure 9 when compared those
2. Different lookup policies perform best depending on whether delay or value deviation is most important to the application. If data quality is more important to the application than cost, and value deviation is more important than delay, simple lookups and piggybacked queries provide the best performance. This can be seen in Figures 4 and 5. In both of these figures, simple lookups and piggybacked queries yield the best quality, other than the "all misses" policy for correlated changes. When value deviation is most important, the expense of taking a cache miss (by not computing an approximate value from neighboring values for these two policies) is worthwhile, since value deviation is deemed most important. If query cost is at a premium compared with quality, using greedy age lookups or greedy distance lookups is preferred. These two policies have the most favorable cost performance in both sensor field models, other than the "all hits" case.

If delay is more important to quality than value deviation, Figures 10 and 11 show that performing greedy age lookups or doing greedy distance lookups yields the best performance. This is true regardless of whether cost or quality is more important to the application. We again assume that the "all hits" case is not useful to realistic applications. For these policies, getting the fast response time of a cache "hit" (which might be approximated from values at one or more neighboring locations) is worthwhile, since low delay is more important than a more accurate value.

The fact that different lookup policies perform best for different application requirements can be explained by examining the underlying delays and value deviations of the policies themselves. For example, consider the case where $A = 0.1$ and changes to the environment are driven by the lab traces. A value of $A = 0.1$ biases quality toward value deviation performance rather than delay performance. In this case, value deviation performance is significantly better when using precise lookups and queries, as shown in Figure 8. This is because the variation in sensor field values is much greater in the Intel Berkeley trace data than values that are drawn from our normal distribution with a time-dependent mean.

### Table 2. Hit ratios, Costs, and Delays for $T = 90$, 0.9 Queries per second, and Trace-driven changes over 54 locations.

<table>
<thead>
<tr>
<th>Policies</th>
<th>Hit ratio</th>
<th>Cost</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>All hits</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>All misses</td>
<td>0</td>
<td>19</td>
<td>4.4</td>
</tr>
<tr>
<td>Simple lookup</td>
<td>0.59</td>
<td>7.7</td>
<td>1.0</td>
</tr>
<tr>
<td>Greedy age lookup</td>
<td>0.78</td>
<td>4.0</td>
<td>0.47</td>
</tr>
<tr>
<td>Greedy distance lookup</td>
<td>0.76</td>
<td>4.4</td>
<td>0.55</td>
</tr>
<tr>
<td>Median-of-3 lookup</td>
<td>0.71</td>
<td>5.4</td>
<td>0.68</td>
</tr>
<tr>
<td>Piggyback queries</td>
<td>0.59</td>
<td>7.7</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Now consider the case where $A = 0.9$ and changes to the environment are again driven by the lab trace data. A value of $A = 0.9$ biases quality toward delay performance rather than value deviation performance. In this case, both delay and cost performance are best for approximate lookups and queries, as shown in Table 2. Figure 11 thus shows that the query cost incurred for doing greedy age lookups or greedy distance lookups is superior to (i.e., less than) the query cost incurred by the other policies for quality that is also better.

It is helpful to summarize the cost and quality performance results presented above as follows:

- When value deviation is more important to quality than delay, there is a linear cost vs. quality trade-off. We obtain the best cost performance by implementing policies that approximate sensor values by using cached values from nearby locations. The best quality performance is achieved by policies that always query and cache the sensor field location specified in the user query.
- When delay is more important than value deviation, policies that approximate values using cached values from nearby locations provide the best cost performance as well as the best quality performance.
- These results hold for both simulated changes to the environment and trace-driven changes to the environment.

### 4 Performance Trends when Value Deviation is Most Important

The results in the previous section provide a thorough understanding of cost and quality performance in pervasive sensing systems for two models of how the environment changes. We next investigate performance trends that emerge as the query rate increases or decreases relative to the rate at which the environment changes.

Because our simulator fully models an environment with correlated changes, we explicitly vary the rate at which the environment changes. To do this we increase the number of locations changed per second from 9 to 90 to 900 while maintaining a query workload with an average rate of 90 queries per second. The Intel Berkeley lab traces specify how the environment changes for our trace-driven experiments. In this case, we vary the relative rate at which the environment changes by decreasing the average query rate from 90 to 9 to 0.9 queries per second.

We begin this investigation by considering cost and quality performance when value deviation is more important in determining data quality than system end-to-end delay. Results for these simulations appear in this section. In the following section, we explore sensor network applications for
which system delay is more important than value deviation.

The same high-level conclusions presented in the last section hold true when the relative rate at which the environment changes is increased by two orders of magnitude. However, some of the underlying results differ either qualitatively or quantitatively. Figures 12 through 17 show these results for correlated changes to the environment. Figures 18 through 23 show the corresponding results for trace-driven changes to the environment.

If value deviation is more important than delay, simple lookups and piggybacked queries again provide the best performance when quality is more important than cost. This can be seen in Figures 12, 13 and 14. If cost is at a premium compared with quality, greedy age lookups or greedy distance lookups trade worse quality for lower cost. In fact, these policies have the most favorable cost performance, other than the “all hits” case. These figures also show that our results are robust with respect to how rapidly our correlated environment changes. For example, the trade-off between cost and quality is linear in Figures 12, 13, and 14. Excluding the “all hits” case again, the range of quality values decreases as the rate at which the environment changes increases. In spite of these differences in quality performance, the relative cost and quality performance of most of the policies remain the same. Figures 15, 16, and 17 confirm a strong positive correlation between value deviation and quality when \( A = 0.1 \). Examining these figures in increasing order illustrates how value deviation increases as the rate at which the environment changes increases.
For trace-driven changes to the environment, simple lookups and piggybacked queries also provide the best performance when quality is more important than cost. This can be seen in Figures 18, 19, and 20. The trade-off between cost and quality is also linear in these figures. Although the costs increase from Figure 18 to Figure 19 to Figure 20, the range of normalized quality is approximately the same (again, excluding the “all hits” case).

Figures 21, 22, and 23 confirm a positive correlation between value deviation and quality when $A = 0.1$ and changes to the environment are driven by our trace data. Examining these figures in increasing order illustrates how value deviation increases as the average query rate (and thus the cache hit ratio) decreases.

5 Performance Trends when End-to-End Delay is Most Important

In the previous section we presented performance results that are important to sensor network applications for which high accuracy (i.e., low value deviation) is the most important factor in the quality of their data. In this section we consider how performance varies as the query rate increases or decreases relative to the rate at which the environment changes when system end-to-end delay is most important. For correlated changes to the environment, we again vary the rate at which the environment changes from 9 to 90 to 900 locations per second while maintaining a query workload with an average rate of 90 queries per second. For the Intel Berkeley lab traces, we again vary the relative rate at which the environment changes by decreasing the average query rate from 90 to 9 to 0.9 queries per second.

We examine system configurations for which delay is a more important component of quality than value deviation. Figures 24 through 29 show such configurations for correlated changes to the environment. For correlated changes to the environment, the cost remains constant for policies as the rate of change for sensor values in the environment increases from 9 locations/s to 900 locations/s. At the same time, the quality performance for all policies is very similar, but not identical. Figures 30 through 32 show configurations for trace-driven changes to the environment. In these system configurations, the cost increases as the average query rate and the cache hit ratio both decrease.
greedy distance lookups yields the best cost performance, and the best quality performance, excluding the “all hits” case. These figures also show that our results are robust with respect to how rapidly our correlated sensor field values change, and that the relative performance differences among our policies remain the same. Similar results hold for trace-driven changes to the environment. These can be seen in Figures 30, 31, and 32. However, it appears that as higher cache hit ratios (e.g., in Figure 30) cause a performance convergence among the caching policies, the median-of-3 policy exhibits better cost and quality performance than doing greedy lookups.

We have seen that for correlated changes to the environment, one or more of the greedy policies provide the best cost and quality performance among the realistic policies. For trace-driven changes to the environment, greedy policies (including the median-of-3 policy) also provide the best cost performance and the best quality performance. This can be seen in Figures 30, 31, and 32, in which there is no trade-off between cost and quality. As costs increase from Figure 30 to Figure 31 to Figure 32, the range of quality values also increases. However, these quality values remain in the continuum between the “All hits” and “All misses” quality values.

6 Conclusion

The following are the contributions of this paper:

- Sensor network caching and lookup policies that improve data quality and query cost. We measure the benefit and cost of seven different caching and lookup policies as a function of the application quality requirements. We show that for some quality requirements (i.e., when delay drives data quality), policies that emulate cache hits by computing and returning approximate values for sensor data yield a simultaneous quality improvement and cost savings. This win-win is because when delay is sufficiently important, the benefit to both query cost and data quality achieved by using approximate values outweighs the negative impact on quality due to the approximation.

- Form and magnitude of the cost vs. quality trade-off. For our seven caching and lookup policies, five of these policies age and then delete cache entries uniformly based on an age threshold parameter, \( T \). We observe that in many system configurations these five policies expose a linear cost vs. quality improvement and cost savings. This linearity is present, we find that the underlying cost vs. accuracy and/or cost vs. delay functions are also linear.

- Bounded cost for some caching and lookup policies. For sensing applications that require bounded resource consumption, we identify a class of policies for which the sensor field

---

**Figure 27.** Delay vs. Quality for \( A = 0.9 \) and 9 of 1000 Correlated changes / second.

**Figure 28.** Delay vs. Quality for \( A = 0.9 \) and 90 of 1000 Correlated changes / second.

**Figure 29.** Delay vs. Quality for \( A = 0.9 \) and 900 of 1000 Correlated changes / second.

**Figure 30.** Cost vs. Quality for \( A = 0.9 \), 90 Queries / second and Trace-driven changes over 54 locations.

**Figure 31.** Cost vs. Quality for \( A = 0.9 \), 9 Queries / second and Trace-driven changes over 54 locations.

**Figure 32.** Cost vs. Quality for \( A = 0.9 \), 0.9 Queries / second and Trace-driven changes over 54 locations.
query rate can be bounded when servicing an arbitrary workload of user queries. Recall that the domain for our user queries is the set of discrete locations in the sensor field. This upper bound is a function of two variables: (1) the number of locations in each sensor field (these locations are also used to index the cache) and; (2) the age threshold parameter, $T$.

- Impact of the manner in which the environment changes on query cost and data quality performance. Our results characterize and quantify cost and quality performance for two different sensing systems, which each monitor environments with different characteristics. These results show that the form and magnitude of the cost and quality performance change, however, the performance trends generally remain the same. Specifically, the performance differences between policies change, but the policies that provide the best quality (and cost) performance in different sensing system configurations are almost always the same.

References


