Optimizing Notifications of Subscription-Based Forecast Queries

Ulrike Fischer[1], Matthias Boehm[1⋆], Wolfgang Lehner[1], Torben Bach Pedersen[2]

[1] Dresden University of Technology, Database Technology Group, Dresden, Germany
{ulrike.fischer,matthias.boehm,wolfgang.lehner}@tu-dresden.de
[2] Aalborg University, Center for Data-intensive Systems, Aalborg, Denmark
tbp@cs.aau.dk

Abstract. Integrating sophisticated statistical methods into database management systems is gaining more and more attention in research and industry. One important statistical method is time series forecasting, which is crucial for decision management in many domains. In this context, previous work addressed the processing of ad-hoc and recurring forecast queries. In contrast, we focus on subscription-based forecast queries that arise when an application (subscriber) continuously requires forecast values for further processing. Forecast queries exhibit the unique characteristic that the underlying forecast model is updated with each new actual value and better forecast values might be available. However, (re-)sending new forecast values to the subscriber for every new value is infeasible because this can cause significant overhead at the subscriber side. The subscriber therefore wishes to be notified only when forecast values have changed relevant to the application. In this paper, we reduce the costs of the subscriber by optimizing the notifications sent to the subscriber, i.e., by balancing the number of notifications and the notification length. We introduce a generic cost model to capture arbitrary subscriber cost functions and discuss different optimization approaches that reduce the subscriber costs while ensuring constrained forecast values deviations. Our experimental evaluation on real datasets shows the validity of our approach with low computational costs.

1 Introduction

Empirically collected data constitutes time series in many domains, e.g., sales per month or energy demand per minute. *Forecasting* is often applied on these time series in order to support important decision-making processes. Sophisticated forecasts require the specification of a stochastic model that captures the dependency of future on past observations. We will refer to such models as *forecast models*. The creation of forecast models is typically computationally expensive, often involving numerical optimization schemes to estimate model parameters. Once a model is created and parameters are estimated, it can efficiently be used to forecast future values, where the *forecast horizon* denotes the

⋆ The author is currently visiting IBM Almaden Research Center, San Jose, CA, USA.

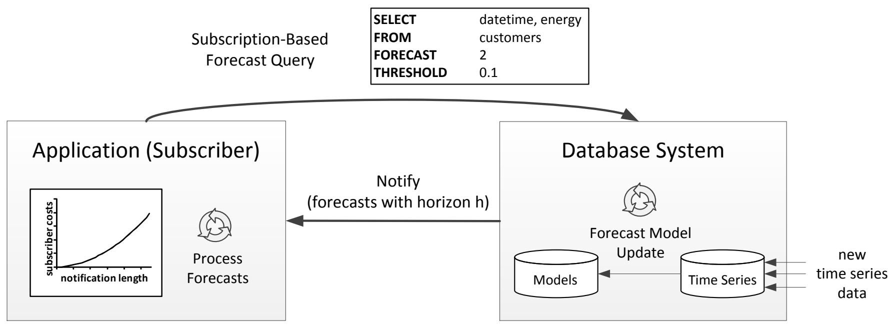


Fig. 1. Overview Subscription-Based Forecast Queries.

number of requested forecast values. As new data arrives, the forecast model requires *maintenance* and improved forecast values might be obtained.

Integrating advanced statistical methods into database management systems is getting more and more attention [1]. These approaches allow for improved performance and additional functionality inside a DBMS. Research on integrating time series forecasting has mainly focused on accuracy and efficiency of ad-hoc [7] and recurring forecast queries [8, 9]. However, applications might continuously require forecast values in order to do further processing. Forecast queries incorporate the unique characteristic that they can provide an arbitrary number of forecast values. However, with each new actual value the underlying model is updated and better forecasts might be available. A dependent application could obtain these values by repeatedly polling from the database. This is very inefficient if forecasts have changed only marginally, especially if the application executes a computational expensive algorithm based on the received forecasts.

In order to tackle this problem, we introduce the concept of *subscription-based forecast queries* that may be seen as continuous forecast queries associated with constraints guiding notifications to the subscriber. A general overview of our approach is shown in Figure 1. A subscription-based forecast query registers at the database system given various parameters, e.g., the *time series* to forecast, a *continuous forecast horizon* and a *threshold* of acceptable forecast deviations. For example, the simple forecast query in Figure 1 requests forecasts of customer energy demand for the next two steps (two forecast values) with a threshold of 10%. The database system itself stores time series data as well as associated forecast models and automatically manages these models [8]. As time proceeds and new time series values arrive models are updated and optionally maintained by the DBMS. This results in better forecast values leading to notifications sent to the subscriber that contain at least the forecast horizon specified by the subscriber (further denoted as *notification length*). For our example query, a notification needs to be sent if new forecasts are available that deviate by more than 10% from old forecasts sent before. The subscriber processes all notifications, where the processing costs of the subscriber often depend

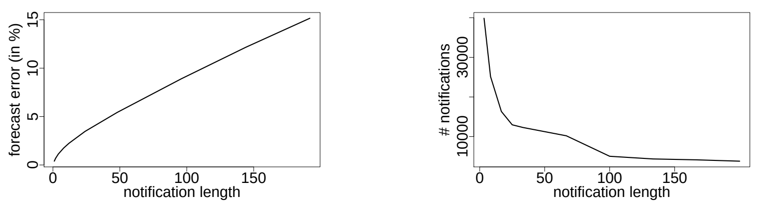


Fig. 2. Influence of Notification Length.

on the notification length. These *subscriber costs* can be communicated to the database system to optimize future notifications.

One important use case of notification-based forecast queries can be found in the energy domain. Forecasting is crucial for modern energy data managements (EDM) systems to plan renewable energy resources and energy demand of customers. In this use case, the EDM system subscribes at a forecasting component, e.g., the DBMS, to receive forecasts on a regular basis. These forecasts are used to balance demand and supply and are crucial to reduce penalties paid for any kind of imbalances, i.e., remaining differences between demand and supply [2, 13, 17]. It is important to update the EDM system just with significant new forecast values to ease the computational expensive job of energy balancing.

Therefore, our objective in this paper is the reduction of the processing costs of the subscriber by trading the number of notifications against the notification length. We distinguish two extreme cases (Figure 2). On the one hand, we can choose a short notification length (i.e., a short forecast horizon). With this approach, we achieve a low forecast error, but we need to send notifications more frequently (as we deal with continuous queries). On the other hand, we can choose a large notification length, resulting in much less notifications, but in a high error for forecasts far from current time. Both extreme cases result in a high overhead at the subscriber side. This first one requires repeatedly processing new notifications, the second one needs to process very long notifications as well as reprocess many notifications containing improved forecasts. To solve this problem, we need to increase the notification length just as far as to ensure accurate forecasts that require a small number of notifications. This poses an important but challenging optimization problem of reducing the costs of the subscriber.

Contributions and Outline Our primary contribution is the introduction of the concept of subscription-based forecast queries, which include a parameter definition, processing model, cost model and optimization objective (Section 2). Then, in Section 3, we propose different computation approaches to minimize the costs of the subscriber according to our optimization objectives defined in Section 2. Our experimental study (Section 4) investigates the performance of the computation approaches, the influence of subscription parameters, the computational costs of our approach as well as the validity of the cost model using real-world datasets. Finally, we survey related work in Section 5 and conclude in Section 6.

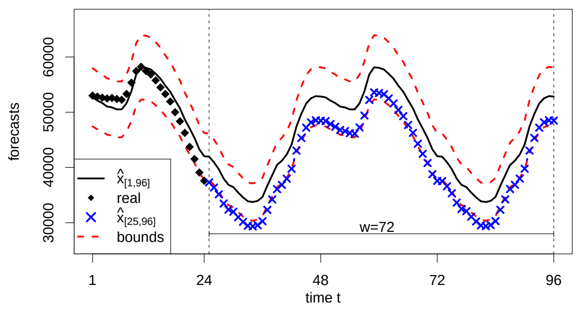


Fig. 3. Development of Forecast Values.

2 Foundations of Subscription-Based Forecast Queries

In this section, we set the foundations of subscription-based forecast queries. We start with discussing the parameters of such a query (Section 2.1), followed by introducing our general processing model that leads to two different kind of notifications to the subscriber (Section 2.2). We then explain our cost model, which captures the costs of the subscriber depending on the number of notifications and the notification length (Section 2.3). Finally, we sketch out our optimization objective of reducing the overall subscriber costs (Section 2.4).

2.1 Forecast-Based Subscriptions

We start with defining the parameters of subscription-based forecast queries.

Definition 1 (Forecast-Based Subscriptions) *A forecast-based subscription* $S = (X, h, \alpha, w, g, k_{max})$ *consists of a time series description* X*, a minimum continuous forecast horizon* h*, a threshold* α*, an aggregation window* w*, an aggregation function* g *and a maximum horizon extension* k_{max}*.*

The time series description $X = x_{[1,t]}$ can be an arbitrary SQL query that specifies the time series to forecast [8]. The continuous forecast horizon h specifies the minimum number of forecast values $\hat{x}_{[t+1,t+h]}$ and implies that at each time t, the subscriber holds at least h forecast values. In addition, each subscription specifies a relative *threshold* α. At time t, we must notify the subscriber with new forecast values if these new forecast values $\hat{x}_{[t+1,t+w]}$ deviate more than α from the old values sent to the subscriber, using a window w and aggregation function g. Examples for aggregation functions are mean (average deviation in window above threshold) or max (maximum in window above threshold), where the decision depends on the intended sensitivity to deviations.

Example 1 *Figure 3 shows a real-dataset example. A forecast model was created up to time* $t = 0$*, using the triple seasonal exponential smoothing model [19]. The solid line displays forecast values* $\hat{x}_{[1,96]}$ *created at time* $t = 0$ *with a horizon of*

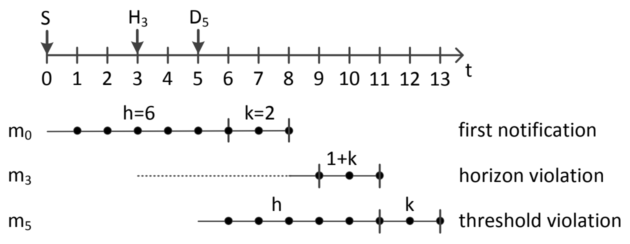


Fig. 4. Processing Forecast-based Subscriptions.

$h = 96$. At each time step $t + i$ with $i > 0$ new real data arrives and we can update the forecast model. At time $t = 24$, we can create new forecast values $\hat{x}_{[25,96]}$ that capture the history better and now slightly differ from the original ones (line with crosses). A subscriber wants to be notified whenever new forecast values deviate by more than 10% from the old ones (dashed lines). We see that at time $t = 24$ some forecast values $\hat{x}_{[25,96]}$ are outside this threshold. Using an aggregation window of $w = 72$ and the aggregation function max we would need to send a notification.

Finally, the maximum horizon extensions k_{max} is specific to our processing model and thus explained in the following.

2.2 Processing Model

At subscription registration time, we send at least h forecast values to the subscriber. From that point in time, notifications are caused by one of the following two reasons. First, new forecast values are sent whenever the subscriber has less than h forecast values (*horizon violation H_t*), where we need to send at least the missing values. However, we can additionally send k values—the *horizon extension*—in order to avoid a lot of horizon violations. There, the subscriber specifies the maximum number of additional forecast values k_{max}. Second, we send a notification if the threshold is violated at time t (*threshold violation D_t*). In this case, we consider *all* sent forecast values as invalid and we resend forecast values with a horizon h plus the horizon extension k. A different approach might be to resend only values that violate the threshold. However, this might lead to systematic errors since forecast values are often based on each other.

Example 2 *Figure 4 shows an example subscription. At time $t = 0$, we create a subscription $S = (X, h, \alpha, w, g, k_{max})$ with minimum horizon $h = 6$. The horizon extension is set to $k = 2$. Initially, we send a notification m_0 with $h + k = 6 + 2 = 8$ forecast values. At time $t = 3$, the subscriber has only $h - 1 = 5$ forecast values (horizon violation H_3). Hence, we send a notification m_3 with $1 + k = 3$ forecast values. We do not override sent values as these are still valid (below the subscription threshold) but send missing values (one value plus $k = 2$ values). Then, at time $t = 5$, the subscriber threshold α is violated according to* *the aggregation window w and function g (threshold violation D_5). We send a notification m_5 with $h + k = 8$ new values, which override all sent values.*

We define the total number of sent values $h + k$ or $k + 1$ as *notification length*. The parameter k has high influence on the number of notifications and individual notification lengths and therefore on the subscriber costs. If we set k quite low, we send many notifications because the horizon is violated. Thus, the subscriber needs to repeatedly process new forecast values leading to high costs. If we set k quite high, the notification length increases and we need to resend a lot of values if the threshold is violated. Thus, the subscriber needs to reprocess many updated forecast values. In the next subsection, we introduce a cost model that allows the quantification of this influence.

2.3 Cost Model

The subscriber cost function can be arbitrary and might be unknown. We therefore use this as a black box function, where we can retrieve the costs for arbitrary notification lengths. Internally, this might be a known analytical function or existing techniques might be used to learn these costs online [18]. However, the costs might be different for $h + k$ new values (threshold violation) or $k + 1$ additional values (horizon violation). Depending on the horizon h and the horizon extension k, we distinguish two cost functions. First, F_C denotes the costs of a complete restart of the subscriber algorithm as necessary for threshold violations, where all forecast values are resent. Second, F_I denotes the costs of an incremental version of the subscriber algorithm that is used for horizon violations, where the subscriber only processes additional values.

These considerations lead to our cost model that calculates the costs between two successive threshold violations D_t and D_{t+i}:

Definition 2 (Threshold Violation Costs) *Assume a horizon extension k. The costs in a threshold violation interval $\Delta D = (D_t, D_{t+i})$ are defined as:*

$$C_{k,\Delta D} = F_C(h + k) + \left\lfloor \frac{\Delta D}{k + 1} \right\rfloor \cdot F_I(k + 1). \tag{1}$$

As explained before, whenever a threshold violation D_t occurs, we send $h+k$ values leading to complete costs of $F_C(h + k)$. Additionally, a certain number of horizon violations occur until the next threshold violation D_{t+i}, each leading to incremental costs of $F_I(k + 1)$. The number of horizon violations equals the number of times $k + 1$ fits in the threshold violation interval ΔD as we send notifications until the end of the interval. If $k + 1$ is smaller or equal than ΔD no additional incremental costs occur.

Example 3 *Consider again Example 2, the first threshold violation occurs at D_5, so according to our definition $\Delta D = 4$. In this interval, we require once complete costs $F_C(h + k)$ at the start of the interval and once ($\lfloor 4/(2 + 1) \rfloor = 1$) incremental costs $F_I(k)$ until the threshold violation D_5. At D_5 a new threshold violation intervals begins, which again starts with complete costs.*

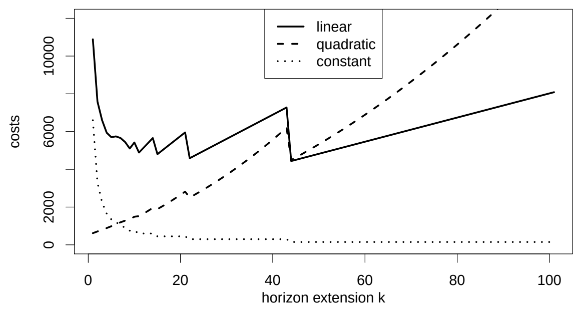


Fig. 5. Example Cost Functions.

The total costs $C_{k,\Delta D}$ in one threshold violation interval strongly depend on the subscriber cost functions F_C and F_I.

Example 4 *Figure 5 illustrates the influence of different cost functions. It shows the theoretical costs according to different horizon extensions k for a fixed minimum horizon $h = 24$ and threshold violation interval $\Delta D = 44$. We used three different cost functions: constant (150), linear ($64x + 150$) and quadratic (x^2), where the same cost function is applied for F_C and F_I. For a quadratic cost function, long notifications are very expensive, so it is best to send many small notifications ($k = 0$). If the cost function only contains setup cost (constant), the goal is to reduce the number of notifications. Hence, we would choose the highest possible horizon extension in order to avoid any horizon violation. The linear function shows a possible cost function between these two extremes. The dips in the linear cost function arise when $k + 1$ is a divisor of ΔD. Thus, all horizon violations fit exactly in the interval and no values are sent unnecessary.*

The threshold violation costs formula makes the simplifying assumption that the threshold violation interval ΔD and the horizon extension k are independent of each other. This might not always be the case as k influences the accuracy of the forecast values and thus also the threshold violation interval. However, preliminary experiments have shown that the impact of k on ΔD is very low.

Based on the defined costs of a *single* threshold violation interval, we are now able to calculate the total costs of a *sequence* of threshold violation intervals.

Definition 3 (Total Subscriber Costs) *Assume a sequence of horizon extensions $\{k_1, \ldots, k_n\}$. Then, the* total costs *of a sequence of threshold violation intervals $\{\Delta D_1, \ldots, \Delta D_n\}$ is defined as:*

$$C_{total} = \sum_{i=1}^{n} C_{k_i,\Delta D_i}. \tag{2}$$

Thus, the best horizon extension depends on the cost function and frequency of threshold violations, which leads to a hierarchy of optimization problems.

2.4 Optimization Problems

Our general *optimization objective* is to reduce the total subscriber costs with regard to the introduced cost model. Furthermore, our *optimization approach* is to choose the best horizon extension, which can be done for different time granularities (offline-static, offline-dynamic, online). This inherently leads to three different optimization problems. These problems are independent of any computation approach and hence, presented separately. Furthermore, they are complete in the sense that additional problems are conceptual composites of them.

The most coarse-grained problem is to choose a single horizon extension for the whole time series, i.e., the sequence of threshold violation intervals (static).

Optimization Problem 1 (Offline – Static) *Assume a sequence of threshold violation intervals $\{\Delta D_1, \ldots, \Delta D_n\}$ and a maximum horizon extension k_{max}. The objective is to minimize the total subscriber costs by choosing a single horizon extension k:*

$$\phi_1 : \min_{0 \le k \le k_{max}} \sum_{i=1}^{n} C_{k,\Delta D_i}. \tag{3}$$

For some datasets, we observe different average threshold violation intervals at different time intervals, where a single horizon extension might fail. For example, often energy demand during week days can be forecasted more accurate than during weekend days, leading to more threshold violations at the weekend. Hence, the second optimization problem is more fine-grained (dynamic) as it aims to find a sequence of horizon extensions for a sequence of time slices.

Optimization Problem 2 (Offline – Time Slice) *Assume a sequence of time slices $\{\Delta T_1, \ldots, \Delta T_m\}$ and a sequence of threshold intervals $\{\Delta D_1, \ldots, \Delta D_n\}$, where $n \ge m$. Each ΔD_i is assigned to exactly one ΔT_j, where l_j is the total number of ΔD_is in time slice ΔT_j. The objective is to minimize the total subscriber costs by choosing a sequence of horizon extensions $\{k_1, \ldots, k_m\}$:*

$$\phi_2 : \min_{0 \le k_j \le k_{max}} \sum_{j=1}^{m} \sum_{i=1}^{l_j} C_{k_j,\Delta D_i}. \tag{4}$$

Finally, the most-fine-grained optimization problem is an adaptive online formulation.

Optimization Problem 3 (Online) *Assume a history of threshold violation intervals $\{\Delta D_1, \ldots, \Delta D_n\}$, horizon extensions $\{k_1, \ldots, k_n\}$ and related costs. The objective is to minimize the total subscriber costs by choosing the* next *k_{n+1}:*

$$\phi_3 : \min_{0 \le k_{n+1} \le k_{max}} C_{k_{n+1},\Delta D_{n+1}}. \tag{5}$$

As the cost function is given by the subscriber or monitored, we "only" need to determine ΔD in order to calculate the best horizon extension. However, ΔD depends on many aspects, e.g., time series characteristics, model accuracy, forecast horizon or subscription parameters. In reality, we do not know when the next threshold violation occurs. However, we can analyze past threshold violation intervals and use them to predict future threshold violations.

3 Computation Approaches

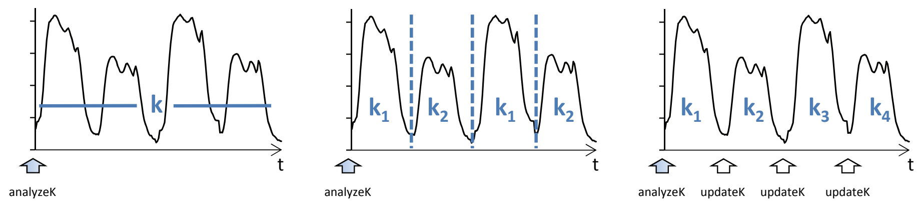
Regarding the defined optimization problems, we now discuss related computation approaches. As a foundation, we first present our general subscription maintenance algorithm (Algorithm 1) as a conceptual framework for arbitrary computation approaches. This includes two major procedures:

First, RegisterSubscriber is called when creating a new subscription S. We first add the new subscription to the list of subscribers on the requested forecast model (line 2). Such a forecast model is stored and maintained for each time series with at least one subscriber. The type of the model (e.g., exponential smoothing) needs to be chosen by a domain expert or by using an heuristic algorithm [10]. To store the list of subscribers itself, we use a simple array structure. However, more advanced data structure can be utilized if the number of subscriber increases (e.g., [3]). Then, we start an analysis phase (line 3). In an offline context, we evaluate single (*static*) or multiple (*dynamic*) horizon extensions. In an online context, we only determine the initial horizon extension. For all three cases, we implicitly interact over a callback interface—implemented by each subscriber—to retrieve the costs F_C and F_I. Finally, we send the first notification to the subscriber with $h + k$ forecast values (line 4).

Second, ProcessInsert is called when a new tuple is added to the time series of the specific model. This requires updating the model to the current state of the time series as well as optional maintenance in form of parameter re-estimation (line 6). For this, we use a simple, but robust, time-based strategy that triggers maintenance after a fixed number of inserts [9]. Then, for each subscriber, we check if the threshold is violated over the aggregation window (lines 8 - 12). If so, we adapt the next horizon extension according to the used strategy (*dynamic*, *online*). Finally, we notify the subscriber with $h + k$ new forecast values. If no threshold violation occurred, we check if the horizon is

Algorithm 1 Forecast Subscription Maintenance

```
Require: $model, subList, currentTime$
 1: procedure RegisterSubscriber($S$)
 2:     $subList \leftarrow$ add($subList, S$)
 3:     analyzeK($S$)
 4:     notify(predict($currentTime$, $S.h + S.k$))
 5: procedure ProcessInsert($newtuple$)
 6:     $model \leftarrow$ maintainModel($newtuple$)
 7:     for $S$ in $subList$ do
 8:         $pmodnew \leftarrow$ predict($S.w$)
 9:         $div \leftarrow$ calculateDiv($pmodnew$, $S.forecasts$)
10:         if $div > S.threshold$ then
11:             updateK($S$)
12:             notify(predict($currentTime$, $S.h + S.k$))
13:         else if isHorizonViolated($S$) then
14:             notify(predict($currentTime + S.h$, $S.k + 1$))
```



(a) Offline – Static (b) Offline – Dynamic (c) Online

Fig. 6. Comparison of Computation Approaches.

violated and we notify the subscriber with the missing values plus the horizon extension (lines 13 - 14). Here, the interaction with the subscriber is also done via the callback interface.

The following computation approaches are based on two observations. First, the best horizon extension in the past can be used to determine the best horizon extension for the future. We therefore propose predictive approaches that analyze the history of threshold violation intervals. Second, there is the problem that threshold violation points strongly fluctuate and we would need to predict the trend of prediction errors. We therefore focus on robust and simple approaches rather than highly dynamic analytical approaches. Figure 6 shows an overview of our computation approaches, which are discussed in the following.

Offline – Static. The first computation approach addresses the coarse-grained Optimization Problem 1, where the objective is to choose a single horizon extension. The static approach determines one horizon extension during the analysis phase of our general algorithm (line 3) and uses this during the whole lifetime of a subscription (Figure 6(a)). To determine the horizon extension, we empirically monitor the threshold violation points over the whole history of the time series. For each tuple in the time series history, we execute an adapted version of the procedure ProcessInsert (Algorithm 1), where we do not notify the subscriber and just monitor whenever a threshold violation occurs. Given the resulting sequence of threshold violation intervals $\{\Delta D_1, \ldots, \Delta D_n\}$, we calculate the costs for different k's using our cost model (Equation 2). This leads to functions such as shown in Figure 5. Finally, we choose the k with minimum costs.

Offline – Time Slices (Dynamic). The second approach solves Optimization Problem 2 to find a sequence of horizon extensions for a sequence of time slices. This approach is beneficial if (1) the time series shows periodic patterns of threshold violation intervals or (2) if the cost function periodically changes over time. During the analysis phase (line 3), we determine a sequence of horizon extensions for periodic time slices (Figure 6(b)). Whenever a threshold violation occurs during execution (line 11), we set the next horizon extension to the horizon extension of the current time slice. The computation approach is similar to the static approach. We just additionally remember in which time slice the threshold violation occurred and compute a separate horizon extension for each time slice. To determine the granularity of the time slices, we either use domain knowledge or we empirically evaluate different types of time slices.

Online Approach. This last computation approach solves Optimization Problem 3 of finding the best next horizon extension. An online approach is beneficial if either the time series model evolves leading to different threshold violation points or the cost function evolves leading to changed costs F_C and F_I. The online approach is repeatedly executed during the lifetime of a subscription to determine the next horizon extensions (Figure 6(c)). At registration time, we need to determine an initial k (line 3). This can be either some predefined parameter or we can use our static approach. The main work is done in the function `updateK` after each threshold violation, where we determine the next horizon extension (line 11) online. Whenever a threshold violation occurs, we monitor the associated ΔD over a predefined window. We then determine the need for recalculating the horizon extension. We analyze two different strategies to trigger recalculation within our experimental evaluation. If recalculation is required, we again compute the costs for different horizon extensions using the monitored sequence of ΔDs and we choose the horizon extension with minimum costs. This requires that the subscriber costs can be retrieved efficiently or we have a (possibly changing) known analytical cost function. Otherwise, we adapt the best k incrementally, i.e., by trying different ks and monitoring the resulting costs.

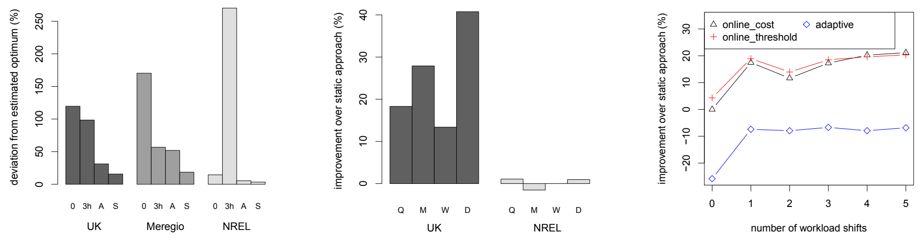
Computational Costs. The costs of all three approaches depend on (1) the time series length n, to evaluate the history of threshold violations, (2) the number of possible horizon extensions m, to retrieve the subscriber costs, and (3) the number of threshold violations d that occur, as the total costs are calculated by the sum over the cost of one threshold violation interval. For the offline approaches, the number of threshold violations d is calculated over the whole history (n represents the whole time series length); for the online approach only the specified window is used (n equals the window size). Following these considerations, the time complexity of all approaches equals $O(n + m \cdot d)$.

4 Experimental Evaluation

We conducted an experimental study on three real world data sets to evaluate (1) the performance (subscriber costs) of our approaches, (2) the influence of subscription parameters, (3) the computational costs of our approach in relation to the subscriber costs and (4) the validity of our cost model.

4.1 Experimental Setting

Test Environment: We implemented a simulation environment using the statistical computing software environment R. It provides efficient built-in forecast



(a) Offline – Static (b) Offline – Time Slice (c) Online

Fig. 7. Performance of Computation Approaches.

methods and parameter estimators, which we used to build the individual forecast models. All experiments were executed on an IBM Blade (Suse Linux, 64bit) with two processors (each a Dual Core Intel Xeon at 2.80 GHz) and 4 GB RAM.

Dataset Descriptions: We used three real-world energy demand and supply datasets. The first dataset (*UK*) includes energy demand of the United Kingdom and is publicly available [14]. It consists of total energy demand data from April 2001 to December 2009 in a 30 min resolution. The second dataset (*MER*) was provided by EnBW, a MIRABEL project partner. It contains energy demand of 86 customers, from November 2009 to June 2010 in a 1 hour resolution. The third dataset (*NREL*) is energy supply data in the form of the publicly available NREL wind integration datasets [15]. It consists of aggregated wind data from 2004 to 2006 in a 10 min resolution.

Forecast Methods: For all datasets, we used triple exponential smoothing with double or triple seasonality as forecast method [19]. This method is an extension of the robust and widely used exponential smoothing and is tailor-made for short-term energy forecasting. We used three seasonalities (daily, weekly and annual) for the UK and NREL datasets but only two seasonalities (daily and weekly) for the MER dataset. We used the first 6 years for UK, 6 months for MER and 2 years for NREL to train the forecast models. The remaining data was used for forecasting and evaluation of our approaches.

4.2 Evaluation of Computation Approaches

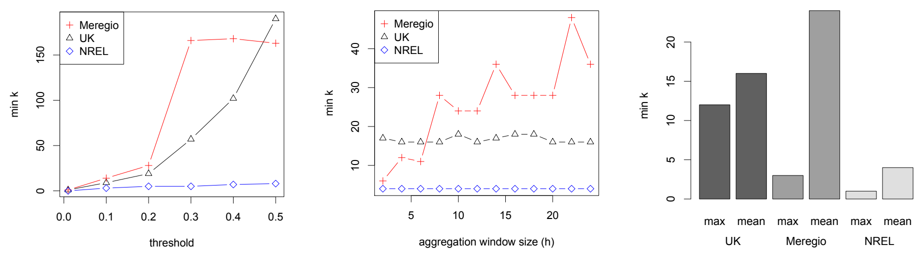
In a first series of experiments, we analyze the performance of our three computation approaches using fixed subscription parameters, i.e., $h = 1\,\text{day}$, $w = 12\,\text{hours}$, $\alpha = 0.15$ and $g = \text{mean}$. For the subscriber cost function, we use the linear function from Example 4.

Offline – Static: We start with comparing our static approach to naïve and adaptive approaches [11]. The first naïve approach never sends more forecast values than requested ($k = 0$). The second naïve approach sends as many forecast values as possible ($k = k_{max}$), where $k_{max} = 3h$. The adaptive approach is independent of the time series history but reacts to notification events. It therefore is a representative of an online approach. Obviously, if a notification occurred due to a horizon violation, the horizon extension was too small. Hence, the adaptive approach increases the horizon extension. In contrast, if a threshold violation occurred, the horizon extension was too high and thus, the horizon extension is decreased. We evaluated different strategies to increase/decrease the horizon extension, where a simple strategy performed best that starts with $k = 1$ and doubles or halves the horizon extension depending on the notification event.

Figure 7(a) shows the result of the four approaches for all three datasets. As all datasets exhibit different total costs, we normalized the cost with the estimated best cost if we would exactly know the threshold violation sequence. We first notice that our static approach (S) always outperforms the two naïve approaches (0 and $3h$) and the adaptive approach (A). The reason is that the static approach includes the subscriber cost function into optimization while the other three approaches act independently from the subscriber costs. In addition, we observe that the datasets exhibit different characteristics leading to different performance of the naïve approaches. For the given subscription parameters, the forecast values of the NREL datasets deviate fast from the old values, leading to a small threshold violation interval on average and to a better performance of small horizon extensions. In contrast, the forecast values of the other two datasets are more accurate leading to larger optimal horizon extensions.

Offline – Dynamic (Time Slices): For our second computation approach we only use the UK and NREL dataset as these datasets are long enough to build meaningful time slices. We analyzed four kinds of time slices: quarter (Q), month (M), weekday/weekend (W), and day (D). Thus, we use different horizon extensions for different time slices, e.g., for every quarter of the year. Figure 7(b) shows the results in terms of the improvement over the static approach by subtracting the estimated best costs and normalizing with the costs of the static approach. For our best case, the UK dataset, all four time slice approaches lead to an improvement over the static approach. We observe the highest improvement for daily time slices, which cover the behavior of customers at different days of a week. The NREL dataset does not contain a typical seasonal behavior. It therefore represents a worst-case example and shows no improvement for all four time slice approaches.

Online Approach: We now relax the assumption of static cost functions and change these functions over time. We use the UK dataset and our linear cost function, where we vary the slope of the cost function [10; 1,000]. Figure 7(c) shows the improvement of the different approaches over the static approach for different numbers of workload shifts. A workload shift switches from the maximum slope to the minimum slope and vice versa. We analyze two different versions of the online approach. The first one recalculates the horizon extension only if the cost function changes (*online_cost*). The second online approach recalculates the horizon extension after every threshold violation (*online_threshold*). Both of our online approaches clearly outperform the adaptive approach that is even worse than the static approach as it acts independently of the subscriber costs. For no workload shifts, the static and online (cost) approach show exactly the same performance as the online approach is never triggered. In contrast, the



(a) Threshold (b) Aggregation Window (c) Aggregation Function

Fig. 8. Influence of Subscription Parameters.

online (threshold) approach slightly improves the static approach. For higher number of workload shifts, we yield high improvements over the static approach because the static approach just determines one horizon extension at the beginning. The online (cost) approach performs very similar to the online (threshold) approach for this use case.

4.3 Influence of Subscription Parameters

In a second series of experiments, we investigate the influence of different subscription parameters , i.e., h, α, w, and g. We again use a linear cost function and set the parameters by default to $h = 1\,$day, $\alpha = 0.15$, $w = 12\,$hours, and $g =$ mean. In the following, we vary one parameter at a time and examine the influence on the best horizon extension k.

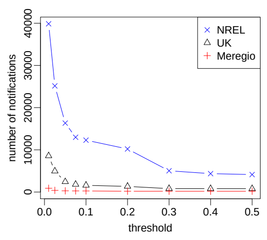
Increasing the threshold α leads to longer threshold violation intervals and thus larger horizon extensions. There is a larger increase for the UK and MER datasets than for the NREL dataset. Both data sets constitute more robust forecast values than NREL leading to few threshold violations with high α.

Figure 8(b) shows the influence of the aggregation window w on the horizon extension. We see a strong increase for the MER dataset, while the horizon extensions for the UK and NREL datasets stay almost constant. This is caused by the customer-level granularity of the MER dataset with many fluctuations. Longer aggregation windows weaken this effect and lead to smaller horizon extensions.

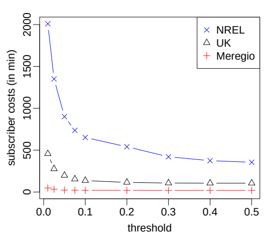
Figure 8(c) shows the best k for the aggregation functions mean and max, where it is consistently larger for mean (with $w > 1$). The UK and NREL dataset show only slight differences, while the MER dataset has a larger difference in the best horizon extension because there forecasts exhibit large fluctuations and the use of max triggers threshold violation notifications immediately.

Note that we do not show an experiment for varying horizons as it does not influence the threshold violation interval and thus the horizon extension.

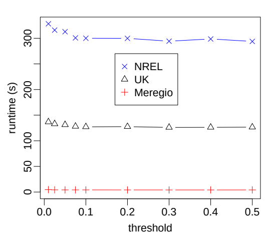
To summarize, the impact of the subscription parameters on k depends strongly on the dataset, which validates our approach of analyzing the time series history to determine the best horizon extension.



(a) Number of Notifications



(b) Subscriber Costs



(c) Runtime

Fig. 9. Computational Costs.

4.4 Computational Costs

In this section, we analyze the computational costs of our approach as well as the overall relationship to the subscriber costs. For this experiment, we use a real cost function from our major use case, the energy domain. This real cost function includes runtime costs of the MIRABEL energy balancing approach [2] and shows a super linear behavior with increasing forecast horizon. We again fix the subscription parameters to $h = 1\,\text{day}$, $w = 12\,\text{hours}$ and $g = \text{mean}$ and vary the subscriber threshold α. To determine the best horizon extension, we use our static approach. Figure 9 shows the trade off between number of sent (or received) notifications, the total resulting subscriber costs and the total time to produce these notifications for all three data sets. Clearly, with increasing threshold α the number of sent notifications decreases as forecasts can have a larger deviation before a notification needs to be sent and thus the notification length increases (Figure 9(a)). In conjunction, the subscribers costs decrease as well as less notifications need to be processed (Figure 9(b)). The runtime of our approach is nearly independent from the number notifications but depends on the data set, i.e., length, granularity and general accuracy (Figure 9(c)). The MER data exhibits the lowest runtime as it consists of hourly data over eight months. In contrast, the NREL data set shows the longest runtime as it is available in a 10 min resolution and hard to forecast. We displayed the overall time to produce all notifications to be comparable with the total subscriber costs. As it can be seen the runtime of our approach is much lower than the total subscriber runtime and thus the subscriber costs form the dominant factor. Note that the average time to produce a single notification equals less than 10 ms for all data sets. This time is measured in a local setting and includes forecast model maintenance, subscription evaluation and sending a notification if required.

4.5 Cost Model Validation

Finally, we validate our cost model, where we use again the real world cost function described in the runtime experiments. To evaluate our cost model we divide the MER data set into two parts of equal size (additionally to the training data

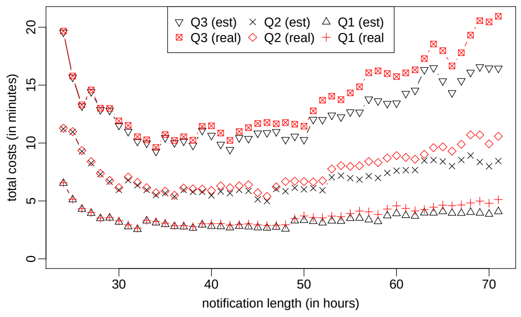


Fig. 10. Cost Model Validation.

used for creating the forecast models). On the first part we estimate the total subscribers costs using our cost model defined in Equation 2. On the second part we measure the real subscribers costs for different notification lengths $h+k$. Figure 10 shows the resulting total subscriber costs for three different kind of queries with increasing complexity (Q1 - Q3), i.e., increasing balancing time. Note that the minimum notification length is determined by the forecast horizon, which is $h = 1\,\text{day}$. For all three queries, our cost model is very accurate for small notification lengths but deviates from the real costs with increasing length. Small notification lengths result in many horizon violations, which are simple to estimate as they are time dependent. With larger notification lengths more threshold violations occurs, where our cost model can never achieve perfect accuracy as we do not know the future. However, most importantly, for all three queries, the minimum of the estimated and real costs are roughly the same leading to the same best horizon extension k. For the UK data set, our cost models performs much better than for the MER data set as this time series, and thus the threshold violation intervals, are easier to predict. In contrast, for the NREL data set our cost model performs slightly worse than for the MER data set as wind data is very fluctuating.

5 Related Work

Existing work has addressed the integration of time series forecasting into DBMS. Approaches to increase speed and accuracy of ad-hoc [7] and recurring queries [8, 9] have been proposed. The Fa system [7] also processes continuous forecast queries. However, in contrast to subscription-based forecast queries no notification conditions can be defined and new forecasts are provided in a regular time interval, independently of the actual changes in forecasts.

The concept of notifying users about incoming events generated by data sources has been intensively investigated in the area of publish-subscribe systems [12, 3]. Probably mostly related to our approach is the work on value-based subscriptions [3]. There, the subscriber wishes to receive an update if a new value (e.g., price) differs from the old one by more than a specified interval. The propose different index structures to scale to a large number of subscribers. In contrast, we need to notify the subscriber with multiple values and our focus is the reduction of the costs of the subscriber.

The problem of tracking a value over time is more generalized in the area of function tracking [5, 20]. An observer monitors a—possibly multi-valued—function and keeps a tracker informed about the current function value(s) within a predefined distance. Yi and Zhang [20] also suggest to use predictions in order to further reduce communication costs. Our work differs in the sense that we already deal with predictions of arbitrary horizons. Hence, in addition to the number of notifications, we reduce the individual notification lengths in terms of the number of forecast values.

The usage of statistical models to reduce communication between two or more entities is applied in a wide range of areas, e.g., sensor networks [6], bounded approximate caching [16] or mobile objects [4]. For example, within sensor networks energy requirements as well as query processing times are reduced by utilizing statistical models in combination with live data acquisition. In the area of bounded approximate caching, cached copies of data are allowed to become out of date according to specified precision constraints. However, all these approaches significantly differ from our problem statement and solutions. First, we consider only time series data, where future values depend on past ones, requiring different statistical models. Second, we use statistical models to estimate future values of the time series instead of missing real values, which leads to specific notification conditions (horizon- and threshold-based) and notification characteristics (resend all values or only additional ones). Finally, instead of reducing the communication costs between the entities, we reduce the costs of applications that process the forecast values.

6 Conclusion and Future Work

We introduced the concept of subscription-based forecast queries. Their main characteristic is a twofold notification condition: horizon- and threshold-based. This results in two different goals of increasing the notification length to avoid horizon-based notifications and reducing the notification length to avoid resending a lot of values if a threshold-based notification occurs. In this paper, we focused on optimizing these notifications to reduce the costs of the subscriber. We developed different computation approaches for different optimization problems, which all use the time series history to determine a suitable notification length. Our experimental evaluation shows the superiority of our computational approaches over alternatives, a significant reduction of the subscriber costs with low computational overhead as well as the validity of our cost in real world situations. In future work, we plan to extend our initial system and discuss data structures and processing approaches to handle a large number of subscribers.

Acknowledgement. This work has been carried out in the MIRABEL project funded by the EU under the grant agreement number 248195.

References

1. M. Akdere, U. Cetintemel, M. Riondato, E. Upfal, and S. Zdonik. The Case for Predictive Database Systems: Opportunities and Challenges. In *CIDR*, 2011.
2. M. Boehm, L. Dannecker, A. Doms, E. Dovgan, B. Filipic, U. Fischer, W. Lehner, T. B. Pedersen, Y. Pitarch, L. Siksnys, and T. Tusar. Data management in the mirabel smart grid system. In *EnDM*, 2012.
3. B. Chandramouli, J. Phillips, and J. Yang. Value-based Notification Conditions in Large-Scale Publish/Subscribe Systems. In *VLDB*, 2007.
4. S. Chen, B. Ooi, and Z. Zhang. An Adaptive Updating Protocol for Reducing Moving Object Database Workload. In *VLDB*, 2010.
5. G. Cormode, S. Muthukrishnan, and K. Yi. Algorithms for Distributed Functional Monitoring. In *SODA*, 2008.
6. A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven Data Acquisition in Sensor Networks. In *VLDB*, 2004.
7. S. Duan and S. Babu. Processing Forecasting Queries. In *VLDB*, 2007.
8. U. Fischer, F. Rosenthal, and W. Lehner. F2db: The flash-forward database system (demo). In *ICDE*, 2012.
9. T. Ge and S. Zdonik. A Skip-list Approach for Efficiently Processing Forecasting Queries. In *VLDB*, 2008.
10. R. J. Hyndman and Y. Khandakar. Automatic Time Series Forecasting: The forecast Package for R. *Journal of Statistical Software*, 2008.
11. P. Levis, N. Patel, D. Culler, and S. Shenker. Trickle: A Self-Regulating Algorithm for Code Propagation and Maintenance in Wireless Sensor Networks. In *NSDI*, 2004.
12. H. Liu and H.-A. Jacobsen. Modeling Uncertainties in Publish/Subscribe Systems. In *ICDE*, 2004.
13. MeRegio Project. 2011. `http://www.meregio.de/en/`.
14. Nationalgrid UK. *Demand Dataset*, 2011. `http://www.nationalgrid.com/uk/Electricity/Data/Demand+Data/`.
15. NREL. *Wind Integration Datasets*, 2011. `http://www.nrel.gov/wind/integrationdatasets/`.
16. C. Olston and J. Widom. Offering a Precision-Performance Tradeoff for Aggregation Queries over Replicated Data. In *VLDB*, 2000.
17. E. Peeters, R. Belhomme, C. Battle, F. Bouffard, S. Karkkainen, D. Six, and M. Hommelberg. Address: Scenarios and Architecture for Active Demand Development in the Smart Grids of the Future. In *CIRED*, 2009.
18. P. Shivam. Active and Accelerated Learning of Cost Models for Optimizing Scientific Applications. In *VLDB*, 2006.
19. J. W. Taylor. Triple Seasonal Methods for Short-Term Electricity Demand Forecasting. *European Journal of Operational Research*, 2009.
20. K. Yi and Q. Zhang. Multi-Dimensional Online Tracking. In *SODA*, 2009.