Ulrike Fischer, Christopher Schildt, Claudio Hartmann, Wolfgang Lehner

Forecasting the data cube. A model configuration advisor for multi-dimensional data sets

Erstveröffentlichung in / First published in:

2013 IEEE 29th International Conference on Data Engineering (ICDE), Brisbane, 08.04.-12.04.2013. IEEE, S. 853-864. ISBN 978-1-4673-4908-6.

DOI: http://dx.doi.org/10.1109/ICDE.2013.6544880

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

Forecasting the Data Cube: A Model Configuration Advisor for Multi-Dimensional Data Sets

Ulrike Fischer, Christopher Schildt, Claudio Hartmann, Wolfgang Lehner

Database Technology Group
Dresden University of Technology
01062 Dresden, Germany
{firstname.lastname}@tu-dresden.de

***Abstract*— Forecasting time series data is crucial in a number of domains such as supply chain management and display advertisement. In these areas, the time series data to forecast is typically organized along multiple dimensions leading to a high number of time series that need to be forecasted. Most current approaches focus only on selection and optimizing a forecast model for a single time series. In this paper, we explore how we can utilize time series at different dimensions to increase forecast accuracy and, optionally, reduce model maintenance overhead. Solving this problem is challenging due to the large space of possibilities and possible high model creation costs. We propose a *model configuration advisor* that automatically determines the best set of models, a *model configuration*, for a given multi-dimensional data set. Our approach is based on a general process that iteratively examines more and more models and simultaneously controls the search space depending on the data set, model type and available hardware. The final model configuration is integrated into F²DB, an extension of PostgreSQL, that processes forecast queries and maintains the configuration as new data arrives. We comprehensively evaluated our approach on real and synthetic data sets. The evaluation shows that our approach significantly increases forecast query accuracy while ensuring low model costs.**

I. Introduction

Currently, a rising trend to integrate advanced analytics into database management systems can be observed [1]. One important and widely used analytical task involves time series forecasting, which is required in many domains, e.g.:

- *Sales forecasting:* Sales forecasts are an essential input to logistics and supply chain management [2], where forecasts are often analyzed and visualized on multiple dimensions involving multiple hierarchies for short-, mid- and long-term planning [3].
- *Smart grid systems:* Smart grid systems, which provide a stable energy supply while accommodating larger amounts of renewable energy, require accurate and real-time forecasts of energy production and consumption over the whole hierarchically organized energy market [1], [4].
- *Display advertisement:* In guaranteed display advertising, where advertisers can buy target user visits on web pages, it is imperative for a publisher to retrieve reliable forecasts of user visits characterized by hundreds of attributes (age, gender, location, etc.) [5], [6].

For all three application areas, the time series data to forecast is typically organized along multiple dimensions and usually stored in data warehouse environments. Along with classical multi-dimensional data cubes, the dimensions provide categorical data, which determine the context of the time series. The time dimension requires special considerations as we are querying and forecasting time series data.

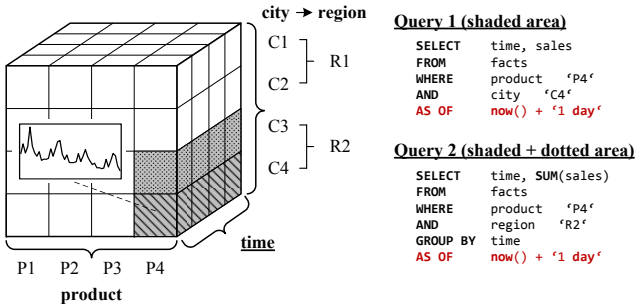


Fig. 1. Forecast Queries in Multi-Dimensional Data Sets

Consider the example of forecasting sales data with sales quantity as measure and time, product, city and region as dimensional attributes (Figure 1). The dimension *time* together with the *measure* forms *time series* for different products and locations. On this data, arbitrary OLAP queries, including *forecast queries* [7], can be submitted. For example, Forecast Query 1 in Figure 1 demands the future of the time series representing the shaded area, i.e., sales forecasts of product $P4$ in city $C4$ over the next day. Forecast Query 2 (shaded and dotted area) requests forecasts of product $P4$ in region $R2$ (containing cities $C3$ and $C4$) and, thus, is an example of forecasting an *aggregated time series*. Further forecast queries are possible, including interactive navigation of forecast results via drill-down or roll-up operations [3].

Most current approaches on processing forecast queries focus on selecting and optimizing a *forecast model* for a single (!) given time series (e.g., [7], [8]). However, additional opportunities arise in the context of multi-dimensional data. To answer Forecast Query 2 of Figure 1, for example, we may exploit various approaches. Most current state of the art would create a forecast model over the aggregated time series (shaded and dotted area). However, alternatively, we might create two forecast models, one over city $C3$ (dotted area) and one over city $C4$ (shaded area), and calculate the forecast of Query 2

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

by aggregating the individual forecasts. We might even use all three models and calculate the forecasts of Query 2 by an arbitrary linear combination of those models. Which approach leads to the highest accuracy?

The best approach strongly depends on the characteristics of the time series data. If data is quite noisy higher aggregation levels will probably lead to higher accuracy [9]. Additionally, we cannot create, store, and maintain a model for each single time series in larger data sets [5], as stored in typical data warehouse scenarios. We need to develop techniques to reuse models for multiple time series while ensuring high accuracy.

Selecting the best set of time series models in multi-dimensional data is challenging. First, we are facing a combinatorial challenge as the number of time series (one for each combination of dimensional attribute values) to forecast might be very high. Second, to judge the benefit and accuracy of a model, we need to build the concerning model [10]. Training time series models is time consuming as parameter estimation of many sophisticated models involves numerical optimization methods that iterate several times over the data [11].

In this paper, we propose an offline model configuration advisor that proposes a configuration of forecast models for forecasting time series in multi-dimensional data cubes. Specifically, we make the following contributions:

- The model configuration advisor is based on an iterative process that adds more and more models to a configuration, allowing the user to retrieve a valid configuration at any time, trading forecast accuracy and model costs.
- We introduce so-called *indicators* that heuristically describe the expected benefit of a model for a time series. Based on these indicators candidate time series are selected, for which a model should be build and examined.
- We developed a control component that regulates the search space in each iteration, based on the data set, model type and hardware platform.
- We introduce F^{2}DB (flash-forward database system) [12] that extends PostgreSQL to natively support and process forecast queries on the final model configuration. F^{2}DB also incrementally maintains models in the configuration as new data arrives.
- We evaluate our approach on real and synthetic data sets showing significantly increased forecast accuracy while ensuring low model costs.

The remainder of this paper is organized as follows: We start by detailing our data model, the type of models we use, the calculation of forecast values as well as the evaluation of a configuration in Section II. An overview of the advisor is given in Section III, followed by a detailed description in Section IV. Then, in Section V, we discuss the integration and management of model configurations in F^{2}DB . We evaluate our approach in Section VI and review related work in Section VII. Finally, we draw conclusions in Section VIII.

II. Preliminaries

We first describe our data model before detailing the forecast models and the calculation of forecast values. We conclude this section by discussing how a configuration of models can be evaluated.

A. Data Model

The data set consists of a set of points in a multi-dimensional space. One of the dimensions is the *time* attribute, one dimension is the *measure* attribute and the remaining dimensions are *categorical* attributes. An ordered sequence of measure values according to the time dimension that have identically values in *all* categorical dimensions form a *base time series* (e.g., Query 1 in Figure 1). Arbitrary aggregations over categorical dimensions are possible and lead to *aggregated time series* (e.g., Query 2 in Figure 1). There might exist functional dependencies between some of the categorical attributes, e.g., city and region. In the remainder of this paper, we only consider SUM as aggregation function, which is most common in our use case scenarios. However, our approach can be easily extended to support other aggregation functions.

Conceptually, we can organize the aggregation possibilities of categorical attributes as a *directed time series hyper graph*, where a node v represents a time series and an edge assigns multiple time series to an aggregated time series (making it a hyper graph). In contrast to the aggregation lattice of the classical data cube [13], this representation focuses on the instance level of the data. Consider again the data set in Figure 1, i.e., a data set with three categorical dimensions—city(C), region(R), and product(P)—with a functional dependency between city and region. Figure 2 shows the graph representation of this data. For the sake of simplicity, we reduced the number of products to two (P_3 and P_4); thus, the graph represents the right half of the cube of Figure 1. The nodes in the lowest level represent base time series, while higher level node stand for aggregated time series. We denoted the aggregated values with a star ($*$). The top node incorporates the aggregation over all time series data and thus represents the total sum over all measures. Note that our data model exhibits a logical model and might be represented differently in an actual implementation. The hyper graph contains three important properties: (1) it is complete in the sense that it contains all aggregation possibilities according to the values of the categorical dimensions, (2) one time series can contribute to several aggregated time series, e.g., the leftmost node $C_1R_1P_3$ can be either aggregated to the node C_1R_1* or to the node $*R_1P_3$ and (3) it explicitly encodes functional dependencies, e.g., C_1*P_3 is not an aggregation possibility of the previous example.

A *query* describes one or several nodes in the hyper graph. For example, Query 1 in Figure 1 references node $C_4R_2P_4$, while the node $*R_2P_4$ is described by Query 2.

B. Forecast Models

Each node v in the hyper graph may potentially be associated with a *forecast model*. A forecast model captures the dependency of future on past data and is created over the time series (either base or aggregated data) of the corresponding node. The forecast method that is used to create the model

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

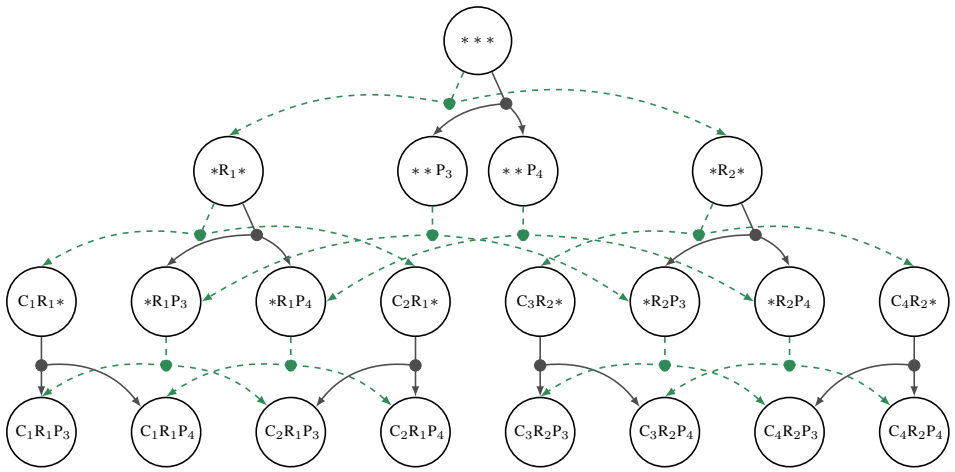


Fig. 2. Graph Representation of Multi-Dimensional Time Series Data

is independent of our approach. However, without conceptual restrictions, we usually employ exponential smoothing and ARIMA models. Both are thoroughly examined (e.g., in [14]), have shown empirically to be able to model a wide range of real world time series [15], and are usually computationally more efficient than elaborate machine learning approaches.

C. Calculation of Forecast Values

A node can utilize any models that exist in the hyper graph (including its own node) to calculate its forecast values. In detail, forecasts of a target node t can be derived from any number of sources nodes S:

$$S \xrightarrow{k} t, \tag{1}$$

where k presents the *derivation weight*. The forecasts of node t are calculated by the sum over the forecast values of S multiplied with the derivation weight k, which is based on the history of source and target time series values.

Let hs_s be the sum over the whole time series history of a node s, the derivation weight $k_{s \to t}$ for deriving t from s can be calculated [16] by

$$k_{s \to t} = \frac{hs_t}{hs_s}. \tag{2}$$

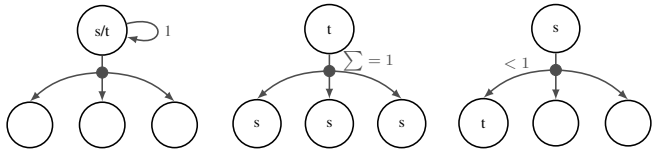
If we have multiple source nodes $S \xrightarrow{k} t$, we sum over the history of all source nodes:

$$k_{S \to t} = \frac{hs_t}{\sum_i hs_s^i}, \tag{3}$$

where hs_s^i represents the time series sum of the i-th node in the vector S.

This weight calculation is based on the assumption that the historical share in the past holds for the future as well. Our calculation approach is built upon the research of Gross and Sohl [16] that empirically analyzed the effect of several approaches on the forecast accuracy.

Examples: Figure 3 shows some intuitive examples of derivation schemes and corresponding derivation weights. First



(a) Direct (b) Aggregation (c) Disaggregation

Fig. 3. Calculation of Forecast Values

of all, a node might utilize the model at its own node (Figure 3 (a)). In this case the derivation weight equals one. This corresponds to most of the state of the art (see Section VII). Second, a node might utilize models at child nodes and retrieve its forecast by aggregation of child forecasts, which corresponds to the classical aggregation operation (Figure 3 (b)). Hereby, the sum over all single node weights equals one. Third, a node might exploit the model of its parent node and apply disaggregation [16] to calculate its forecast (Figure 3 (c)). Hereby, the weight represents the ratio of the child node on the parent node to scale down the forecast of the (aggregated) parent node. Consider our running example to calculate the forecasts of Query 2, we might utilize node $*R_2P_4$ (direct), nodes $C_3R_2P_4$ and $C_4R_2P_4$ (aggregation) as well as node $*R_2*$ or $***$ (disaggregation).

Each derivation scheme influences the accuracy of the resulting forecast values. Studies have shown that aggregation can improve the accuracy over a direct approach [10]. Disaggregation is beneficial if base data is quite noisy [9]. The combination of forecasts can further improve the accuracy [17]. In the following, we call an assignment of models and derivation schemes to nodes a *model configuration*.

D. Configuration Evaluation

The quality of a model configuration can be judged by two measures: *forecast error* and *model costs*.

Forecast error: Our main goal is the minimization of the forecast error of all nodes in the time series graph. First of

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

all, this requires an accuracy measure to calculate the error of a single node. Many measures of forecast accuracy have been proposed in the past [18]. We decided to use the *symmetric mean absolute percentage error* (SMAPE), which is a scale-independent accuracy measure and takes values between 0 and 1, making it easily comparable. It is defined as

$$SMAPE = \frac{|x_t - \hat{x}_t|}{x_t + \hat{x}_t}, \quad (4)$$

where x_t describes the real value of a time series at time t and $\hat{x}_t$ the corresponding forecast value. Calculating a node's error requires the division of the time series into a training part, over which the model is created, and a testing part for the error calculation itself.

The errors of single nodes are combined into one quality measure representing the overall error err of the configuration.

Model costs: Although we create models offline, we need to spend time online to maintain them, which we refer to as *model costs*. Model maintenance is required if time proceeds and new actual values arrive at the nodes in the time series graph. Maintenance involves updating the model to the current state of the time series as well as optional parameter re-estimation. Especially in application domains that require real-time answers (e.g., display advertisement [5]) maintaining a model for each single node might not be possible. Thus, we might be willing to tolerate a configuration with higher forecast error if it implies lower model costs. The maintenance costs of a node depend on the time series characteristics (how strongly fluctuates the time series) as well as the maintenance strategy (when and how often are model parameters required to be reestimated). We therefore utilize the total model creation time over all models in the configuration as worst case approximation if parameters have to be re-estimated after every new value added to the time series.

Based on these two evaluation measures, our model configuration advisor tries to find a model configuration that exhibits a low forecast error while ensuring low model costs.

III. Model Configuration Advisor Overview

We start by giving an overview of our solution and by describing our core concept, so called *indicators*, that describe the expected benefit of a model in a configuration.

A. Overview

The model configuration advisor receives a *time series graph* as input and iteratively outputs a *model configuration* as well as associated *forecast error* and *model costs* (Figure 5). Ideally no further parameterization input should be needed when running the advisor. The execution of the advisor is based on an iterative process that in each iteration adds new models to the configuration and, optionally, removes models. In detail, the advisor consists of the following four phases:

- In the first phase, the *candidate selection* phase, the advisor utilizes several heuristics, so called *indicators*, to determine and rank a set of models that might be added or deleted from the configuration.

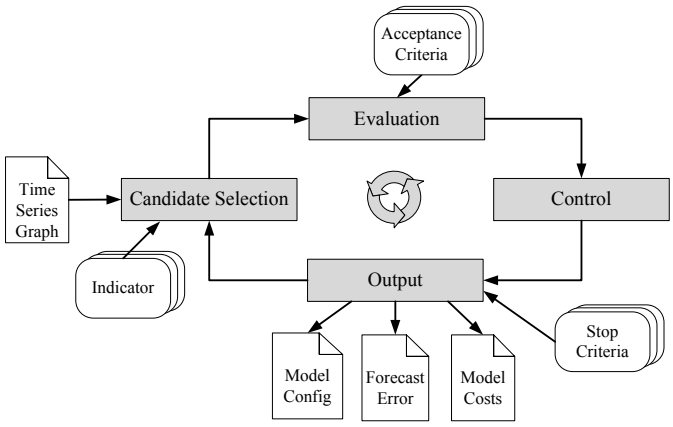


Fig. 5. Configuration Model Advisor Overview

- In the *evaluation* phase, the benefit of adding or deleting candidate models is evaluated and a decision is made utilizing several *acceptance criteria*. In this phase, models for candidates are explicitly created and evaluated.
- In the *control* phase, internal variables are updated as well as parameters are adjusted to optimize further iterations. Parameters are regulated taking data characteristics, model creation time and hardware platform (e.g., number of processors) into account.
- Finally, in the *output* phase, one or several *stop criteria* are evaluated. Hereby, the advisor either continues (and optionally outputs an intermediate configuration) or terminates outputting the final configuration.

This iterative process ensures that the advisor can be canceled at any time, if either (1) the forecast error is acceptable or only shows slight improvements with more models or (2) the maximum acceptable model costs are reached.

B. Indicators

To find model candidates, we use *indicators* that heuristically indicate the expected benefit of a model without removing or, more importantly, building it. The calculation of these indicators should be efficient (in comparison to model creation) and accurate (to correctly reflect the benefit). Obviously, as our indicators are supposed to judge the benefit of a possible model, we can not use any information about the model itself. We can only use heuristics that are based on the available historical time series data. Such heuristics might either focus on a single time series or on the relationship of time series. Especially for large data sets, the accuracy of derivation schemes is very important as we can not build a model for each single time series. To allow such scalability, our indicators are based on the *accuracy of derivation schemes*, i.e., on time series relationships.

Historical Error: First, analogue to other approaches in this area [17], [19], we can use the time series history to evaluate the historical error of a scheme $s \xrightarrow{k} t$. As we do not know a model for node s, we can only assume perfect accuracy and use the real historical values of s as forecast values. Those

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

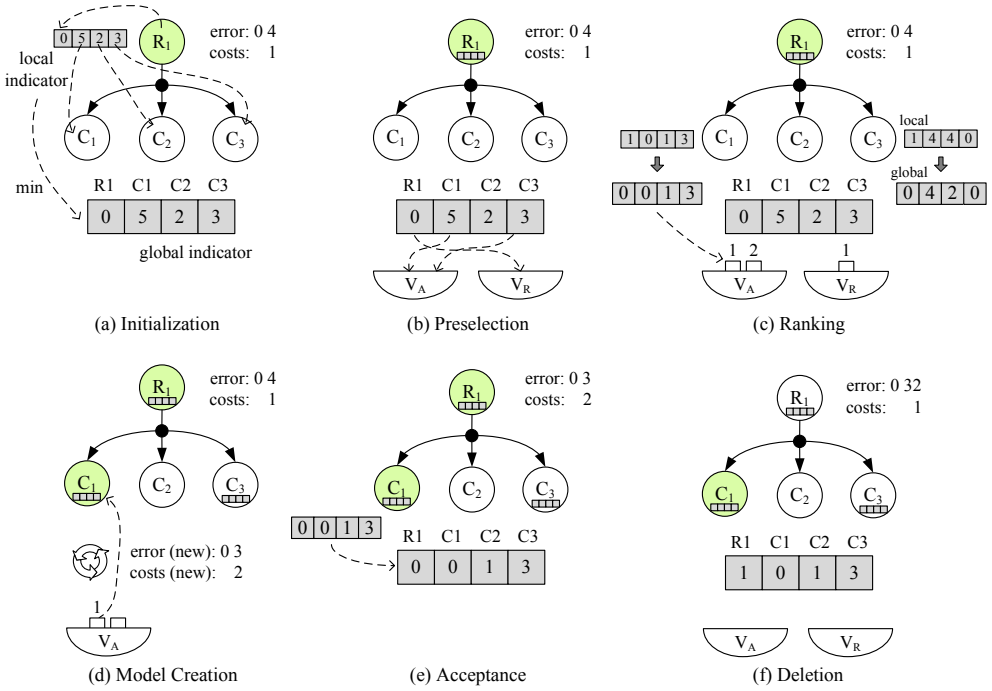


Fig. 4. Example Iteration of the Configuration Advisor

values together with the derivation weight $k_{s \to t}$ compute the forecast values of t, which are compared with the real values of t. In this paper, this error is calculated over the entire history as the time series from our real-world data sets are quite short, but a different history length might also be used.

Similarity: A second important factor in the accuracy of derivation schemes is the similarity of time series [9]. Obviously, a scheme will be quite accurate if the corresponding time series are very similar. However, as as the number of possible derivation schemes might be very high, we need an efficient measure to capture this similarity. One possibility is the analysis of the derivation weights $k_{s \to t}$. Constant weights indicate a high similarity and consistent relationship between nodes s and t. In contrast, if weights strongly fluctuate over time, the corresponding scheme is quite unstable and leads to low accuracy. This stability is captured by computing the variance of the weights over the entire time series history.

Other indicators are possible, but within our experimental evaluation we show that these indicators have a strong correlation with the real forecast error. However, the concrete indicator type is independent from our framework and new indicators can easily be added.

These indicators are combined into a single accuracy measure, where a low indicator value (i.e., low historical error and low variance of weights) corresponds to a high derivation accuracy. A *local indicator* array for a source node s can be created, where each entry indicates the derivation error of target node t from source node s in the scheme $s \xrightarrow{k} t$ (see Figure 4 (a)). The indicator value of a node with itself is zero.

A *global indicator* is retrieved by computing the minimum over all currently existing local indicators. Analogue, the global indicator contains an entry for each node in the graph indicating the minimum error over all derivation schemes.

IV. Model Configuration Advisor Details

We now detail each of the four phases of the advisor. We illustrate each step with an example using a smaller version of the time series graph from Figure 2. It contains four nodes (see Figure 4) – the top node represents aggregated sales in region R_1, while each leave node covers individual sales of a certain city (C_1, C_2, C_3).

A. Candidate Selection Phase

In the candidate selection phase, we utilize the indicators to choose a set of *positive* nodes V_A that might benefit from a model as well as a set of *negative* nodes V_R where a model should be removed (*preselection*). We then examine these candidates more closely by creating new local indicators and *rank* them according to their expected benefit in a configuration.

1) Preselection: In the preselection step, we examine the current status of the configuration, for which we store a local indicator array for all nodes s that currently contain a *forecast model*. A global indicator is created as described in Subsection III-B, which indicates the current status of the configuration.

Example: In our running example, we start with a configuration, where only one model exists for the top node R_1 (Figure 4 (a)). A local indicator is initialized for this node. The global indicator equals exactly the local indicator in the given example, as there are no other local indicators. The values in the top right corner of Figure 4 (a) show the real configuration error and configuration costs (see Subsection II-D).

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

Nodes with high values in the global indicator I probably have a high error and should be chosen as positive candidates V_A. In contrast, nodes with low values should be chosen as negative candidates V_R:

$$V_A = \{v | v \in V, I_v > E(I) + \gamma \cdot \sigma(I)\} \text{ and} \quad (5)$$

$$V_R = \{v | v \in V, I_v = 0\}. \quad (6)$$

We therefore add all nodes above the average of the global indicator $E(I)$ to the set of positive candidates V_A. The parameter γ together with the standard deviation $\sigma(I)$ of the global indicator is used to regulate the number of positive candidates and set in the *control phase*. In our current indicator approach, the indicator value of a node with a forecast model is always zero. We, therefore, add all nodes with an indicator value of zero to the set of negative candidates V_R.

Example: Assume the parameter γ is set to zero. In Figure 4 (b), the two nodes C_1 and C_3 are higher than the average indicator value ($E(I) = 2.5$) and are added to the positive candidates V_A, while the node R_1 with an indicator value of zero is added to the negative candidates V_R.

2) Ranking: Within the ranking step, we examine all positive candidates V_A more closely by creating a local indicator (if not already present) for each candidate node. A new global indicator is created and stored including the respective node. We then rank the set V_A by those global indicators in decreasing order, arranging nodes with the highest benefit first.

Example: Figure 4 (c) visualizes the ranking step. A local indicator was created for the nodes C_1 and C_3 and subsequently for each node a new (temporary) global indicator was computed, which constitutes the minimum over the current global indicator and the local indicator of the corresponding node. The global indicator of node C_1 shows on average lower indicator values than node C_3 and therefore takes first place in the ranked queue of candidate nodes.

Similar, we examine all negative candidates by removing the local node indicator from the current global indicator. We rank the nodes in the set V_R based on these indicators in ascending orders, arranging nodes which lowest benefit to the current configuration first.

B. Evaluation Phase

In the evaluation phase, we evaluate the nodes in the set V_A and V_R to decide whether they should be added to or deleted from the configuration. Hereby, we explicitly *create* forecast models for positive candidates V_A. We then evaluate the *real* forecast error and benefit of a model in the configuration and decide whether a new model should be *accepted* to the configuration as well as if a model should be deleted.

1) Model Creation: From the ranked set of positive candidates V_A, the top n nodes are chosen, for which a model is created. The number of nodes n is restricted by the number of available processors. For creation itself, the time series history is divided into a training part, over which the model is created as well as an evaluation part, over which forecast values are calculated. Creating a forecast model requires estimating its parameters using standard local (e.g., Hill-Climbing) or global (e.g., Simulated Annealing) optimization algorithms (for more details see [11]). Note that a model is always created over a single time series of a node s. The forecast values are then utilized to calculate the effect of this model on the multi-dimensional data cube by computing the accuracy of the model at its own node as well as in derivation schemes. Hereby, each node in the current configuration knows its current best forecast error and associated derivation scheme. If the new model improves the forecast error of a node, it replaces the old error and derivation scheme, leading to a new overall configuration error err_{new}.

Example: In Figure 4 (d), we choose to build a model for the first node in the ranked set V_A. After creating this model, we can compute the new error and costs of the configuration. The model costs have increased as there are now two models in the configuration. In contrast, the forecast error has decreased.

2) Acceptance: To decide whether a model should be accepted to the configuration, we need to compare the new configuration error err_{new} with the previous configuration error err_{old}. A simple *acceptance criteria* might just compare both errors and, if the error is better, the model gets accepted:

$$err_{new} < err_{old}. \quad (7)$$

Additionally, we allow to reject models that achieve only a small error improvement in favor of more beneficial models. A generalized acceptance criteria sets the error improvement in relationship with the overall configuration costs:

$$\alpha\, err_{new} + (1-\alpha)\, cost_{new} < \alpha\, err_{old} + (1-\alpha)\, cost_{old}, \quad (8)$$

where $\alpha \in [0, 1]$ regulates the influence of the model costs. If $\alpha = 1$ the acceptance criteria equals the error-based criteria. The proposed acceptance criteria requires a normalization so that error and costs are comparable.

If a model gets accepted, it is added to the current configuration, the configuration error and costs are updated as well as the global indicator values. If a model is rejected according to the acceptance criteria and additionally does not improve the forecast error, it is marked so that it will not get selected again as positive candidate.

Example: As the new model decreases the configuration error, it is accepted according to the criteria in Equation 7. Subsequently, we update the global indicator value, configuration error and configuration costs (Figure 4 (e)).

Deletion is handled similarly by examining the effect of deleting the top node in the sorted list V_R. A node is deleted if this improves the configuration according the defined acceptance criteria (Equation 8). This approach removes nodes that have been added too greedy to the configuration but do not contribute significantly to the overall accuracy.

Example: In Figure 4 (f) the model at node R_1 has been deleted. While the configurations costs have decreased by one, the configuration error has slightly increased. However, this slightly higher configuration error might be acceptable due to the lower costs. Finally, the global indicator has been updated and includes now just the local indicator array of node C_1.

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

C. Control Phase

The control phase has two main functionalities. First, it regulates the search space and search time by setting the advisor parameters. Secondly, it utilizes the available hardware to search for additional derivation schemes in the background.

1) Parameter Settings: Our advisor utilizes three important parameters, the size of the indicator arrays $|I|$, the number of positive candidates, regulated by γ, and the acceptance threshold α.

Setting $|I|$: Searching the whole space of derivation schemes is time- and memory-consuming and not possible for larger data sets. A smaller indicator size $|I|$ requires less memory and less computing time in the candidate selection phase, but might also reduce the forecast accuracy as less derivation possibilities are considered. Our current strategy considers memory space and restricts the indicator size $|I|$ so that indicators for all nodes fit in memory. The local indicator of a node s is then constructed by including those nodes which are closest to s in the time series graph.

Setting γ: The parameter γ regulates the number of candidates that are chosen in the preselection step. The more candidates are selected, the higher the runtime of the candidate selection step as each candidate requires the creation of a local indicator in the ranking step. However, selecting more candidates might also increase the accuracy of the configuration as we analyze more nodes. Nevertheless, the candidate selection phase should not be more expensive than the evaluation phase, otherwise we could just invest the time to directly create forecast models, which is always the most accurate approach. Based on theses considerations, our strategy to regulate γ takes the indicator creation time as well as the model creation time into account. In detail, we assume a normal distribution of the indicator value and set γ initially so that the number of candidates roughly equals the number of processors on the machine. In each iteration, we compare the time spent in the candidate selection phase with the time spent in the evaluation phase and either increase or decrease γ.

Setting α: The parameter α determines the error improvement a model has to achieve in order to be accepted to the configuration and sets it in relationship with the costs of the model. If α is quite small only models that achieve a high error improvement are accepted, while the higher the parameter α the more models get accepted. With $\alpha = 1$ all models are accepted that lead to an improvement of the configuration, independently of the model costs. Initially, α is set to a low value (usually 0.1) and then continuously increased (until $\alpha = 1$). The parameter α is increased if either (1) a certain number of rejects has occurred, (2) the maximum number of iterations is reached or (3) the error improvement is too small. This approach allows a reasonable runtime of the advisor even for larger data sets as the user can stop the advisor at any point in time, knowing that the most beneficial models are included in the current configuration.

2) Optimizations: The introduced indicators consider only derivation schemes from single source nodes. Although initial experiments have shown that these schemes are most important and have the highest impact on the configuration error, derivation schemes from multiple source nodes might further increase the forecast accuracy. We therefore integrated an additional asynchronous component that addresses this task. It iteratively selects a target node and a random number of source nodes from the time series graph, where the possibility of selecting a source node decreases with increasing distance from the target node. The derivation accuracy of the selected nodes is then evaluated and applied to the configuration if the configuration accuracy is improved.

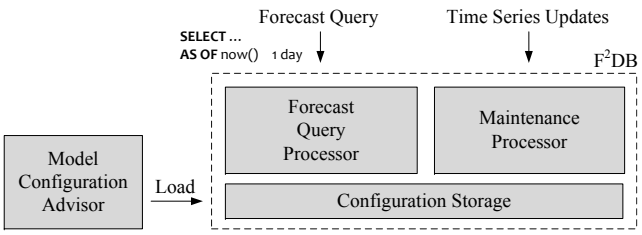


Fig. 6. Integrating Forecasting inside the DBMS

D. Output Phase

The model configuration advisor continuously outputs the forecast error as well as the model costs of the current best configuration. Thus, a user can interrupt the advisor at any point in time if the forecast error is acceptable, shows only slight improvements with more models or the maximum acceptable model costs are reached. Additionally, the user can set predefined *stop criteria* to trigger automatic termination of the advisor. Again, these can be error-based, either by giving an absolute error or relative error with respect to the initial configuration, or cost-based, either by absolute or relative costs. If no stop criteria are given, the advisor will continuously increase the parameter α and stop when $\alpha > 1$ (see Subsection IV-C.1).

Example: After finishing one iteration of the advisor, we retrieve the configuration in Figure 4 (f). We can now continue with a new iteration or we can output this configuration along with its error and costs.

V. Management of Model Configurations

Our initial goal was the optimization of forecast queries in multi-dimensional data sets. Thus, once we have found a model configuration, we need to store, manage, and update it for efficient and accurate query processing. In previous work, we already introduced F²DB, an extension of PostgreSQL, that natively supports and processes forecast queries [12]. Within this section, we shortly discuss the main extensions and implementation issues relevant for this paper. Besides the implementation of the model configuration advisor itself, we need to store the final configuration in the DBMS and use it for the processing of forecast queries and updates to the time series data (Figure 6).

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

Configuration Advisor: We implemented the advisor as a stand alone component—outside of PostgreSQL. It is responsible for building the time series graph, selecting a model configuration as described in the previous two sections, and finally loading the result in the database. The time series graph might be automatically derived from the data (e.g., by foreign key relationships). During execution the advisor utilizes the forecasting capabilities of F^{2}DB to build models for certain nodes in the evaluation phase and retrieve their accuracy.

Configuration Storage: To store a model configuration we utilize the standard relational tables and index structures of PostgreSQL. In detail, we added two tables: the first one stores the time series graph and model configuration (including model assignments, derivation schemes and corresponding weights), and the second table stores the forecast models itself including state and parameter values.

Forecast Query Processor: A forecast query is rewritten to access relevant nodes in the time series graph. It, thus, finds the nodes, loads the necessary models and calculates the forecasts. The processing of a forecast query is very fast as there is no need to access the base tables containing the time series data.

Maintenance Processor: As time proceeds, new time series values are inserted into the database system, requiring maintenance of the models in the model configuration. Maintenance involves updating the state of the models and derivation weights to the current point in time as well as optional parameter reestimation of forecast models. Maintenance of nodes of aggregated time series requires that new time series values for all child nodes are available. To address this issue, we currently batch inserts until a new value is available for each base time series for the next time stamp. We then advance time in the whole time series graph by processing all inserts at once and by updating the state of the models and derivation weights, which can be done incrementally. Optionally, we mark models as invalid if parameter re-estimation is necessary (e.g., based on a time- or threshold-based strategy, see [12] for details). If a query references an invalid model, we trigger immediate parameter reestimation. With this approach we reduce maintenance overhead by delaying parameter reestimation until the model is actually referenced by a query.

VI. Experimental Evaluation

We conducted an experimental study on several real-world and synthetic data sets to evaluate (1) the performance of the advisor with respect to forecast accuracy and model costs and compared to other approaches, (2) the influence of the advisor's parameters as well as (3) the runtime behavior of the advisor and the runtime of forecast queries in F^{2}DB.

A. Experimental Setting

To evaluate our approach, we consider several real-world data sets from different domains as well as synthetic data sets:

Tourism: The *Tourism* data set was already used in several papers in the forecasting literature [17], [20] and contains quarterly observations on the number of visitors for the Australian domestic tourism indicating the tourism activity in Australia. The sample contains 32 base time series from 2004 to 2011 according to two dimensions — purpose of visit (holiday, business, visiting friends and relatives, other) and state [21].

Sales: We obtained an excerpt of sales data from a market research company. The excerpt contains 27 base time series according to different products and countries in a monthly resolution from 2004 to 2009.

Energy: The third real data set originates from the energy domain and was obtained by EnBW during the Meregio project [22]. It contains energy demand of 86 customers, from November 2009 to June 2010, in an hourly resolution.

GenX: Finally, we generated synthetic time series data for a certain number of base time series X. These are then summed to obtain the aggregated data for the levels above. To create the time series graph, we use three levels if $X < 1,000$, four levels for $1,000 \leq X < 10,000$, five levels for $10,000 \leq X < 100,000$ and six levels for $X \geq 100,000$. The time series data itself was generated by a SARIMA process [11] using the statistical computing software environment R.

All data sets where loaded into the F^{2}DB Postgres database. The creation of models for nodes in the time series graph representing aggregated time series requires aggregation of the data in the database. To avoid repeatedly scanning the same data, we initially created all aggregated time series for the whole time series graph. We analyzed different forecast models for all four data sets and found that triple exponential smoothing worked best in most cases, where we set the seasonality according to the granularity of the data. For each time series, we used about 80% of the data to train the forecast models and the remaining data to find and evaluate the best configuration. The following experiments were executed on an AMD Opteron System with 12 cores and 32 GB RAM.

B. Accuracy Analysis

In a first series of experiments, we compare the performance of our approach to several alternatives.

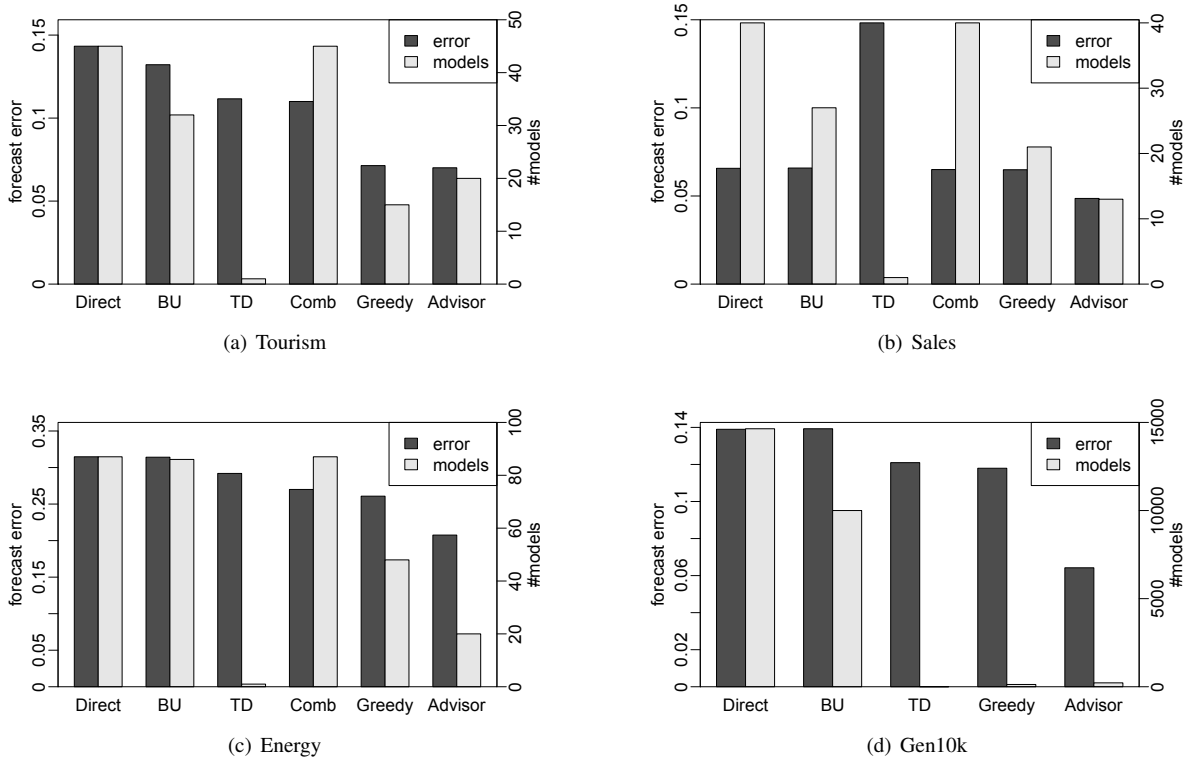
The first group of alternatives are from the area of hierarchical forecasting [23] in the forecasting literature and fix a configuration independent from the data:

Direct: The naive direct approach creates a model for each node in the time series graph and uses the model to directly calculate the forecasts of the corresponding node.

Bottom-Up: Arguably the most commonly applied method in forecasting literature [17] is the *bottom-up approach* (e.g., [10], [24]), where only forecasts for base time series are created and aggregated to produce forecasts for the whole time series graph.

Top-Down: The other commonly applied method in forecasting literature is the *top-down approach* (e.g., [25], [9]), where the most common form distributes the forecasts of the top node down the hierarchy based on the historical proportions of the data [17]. Gross and Sohl [16] analyzed

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880



(a) Tourism

(b) Sales

(c) Energy

(d) Gen10k

Fig. 7. Accuracy Analysis

several versions of this approach, where a simple method that uses the proportions of the historical averages performed best.

Additionally, we consider two alternatives that empirically choose a configuration based on training data:

Combine: Hyndman et. al. [17] proposed a general hierarchical forecasting framework that independently forecasts all series at all aggregation levels of the hierarchy and computes a regression model to optimally combine and reconcile these forecasts. The code is publicly available as part of the hts package of the statistical computing software R.

Greedy: The second empirical approach is a simple greedy approach that initially builds *all* forecast models for all nodes in the graph and then selects in each step the model with the highest benefit with respect to forecast accuracy [19]. It stops when there is no model left that improves the accuracy. To calculate the forecasts, it only considers the traditional derivation schemes aggregation, disaggregation and direct.

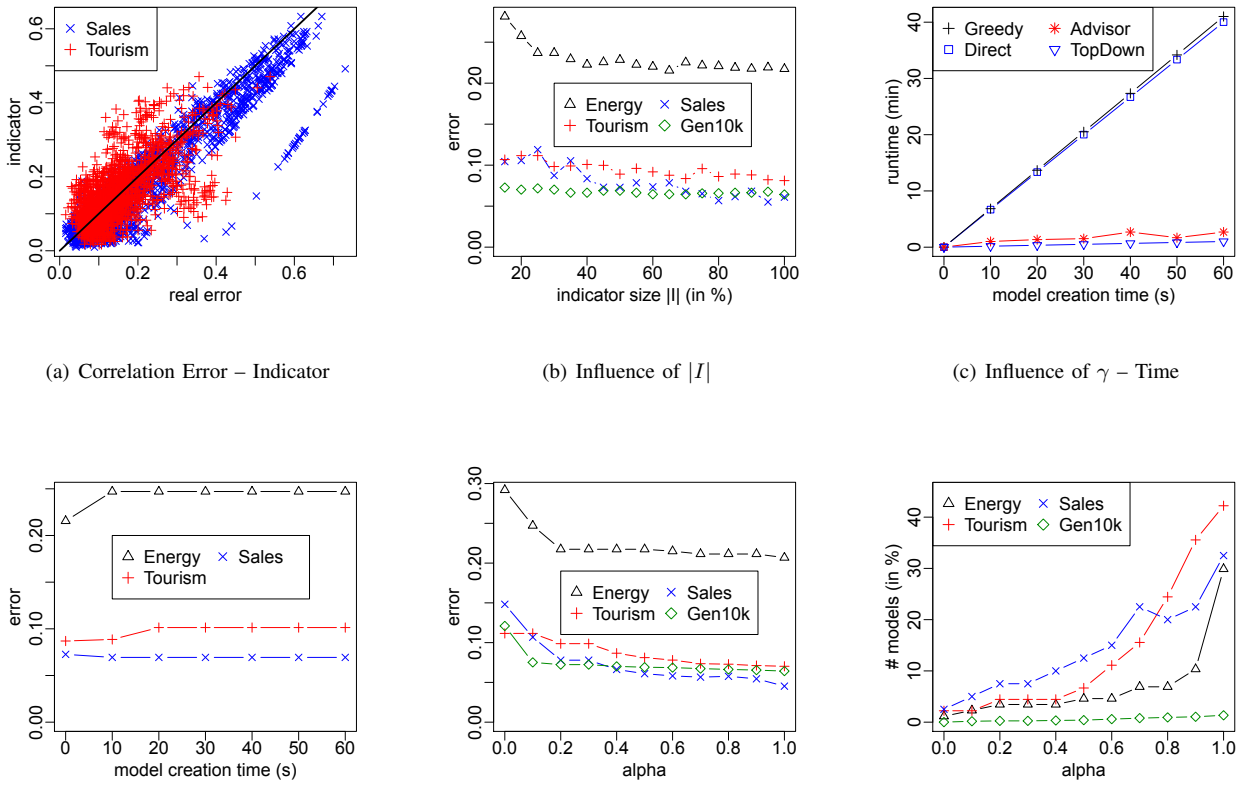
We executed these approaches for each of the real-world data sets and one synthetic data set containing 10,000 base time series $Gen10k$ (Figure 7). The dark grey bars show the forecast error of the corresponding approach (left y-axis) and the light grey bars the number of models in the final configuration representing the model costs (right y-axis). For all data sets, our advisor results in the lowest overall forecast error and outperforms the majority of the other approaches in model costs. The data-independent approaches – direct, bottom-up (BU) and top-down (TD) – show different behaviors for different data sets. For $Tourism$ and $Gen10k$, the top-down approach exhibits the best forecast error, while for the $Sales$ data set, the direct and bottom-approach show a lower error. In contrast, for the $Energy$ data set, all approaches behave equally with respect to the forecast error. This proves our initial statement that the best approach depends on the data set. In terms of model costs, the top-down approach exhibits the lowest costs as only one model is created, while the direct approach shows the highest costs as models for all nodes are required. The Combine approach (Comb) shows a slightly lower forecast error than the data-independent approaches for all three real-world data sets, but requires maximum model costs as models for all nodes are created and utilized. We did not execute the Combine approach for the $Syn10k$ data set due to the long execution time ($>$ one day). The Greedy approach outperforms all other approaches in terms of forecast accuracy and models costs, but is beaten by our model configuration advisor. By using a generalized derivation scheme and carefully selecting most beneficial models first, the model configuration advisor achieves the lowest forecast error while keeping low model costs.

C. Parameter Analysis

In a second series of experiments, we more closely analyze the different parameters of the model configuration advisor.

Indicator Accuracy: First of all, we investigated the applicability of our two indicators. For this, we analyzed the correlation between the indicators and the real forecast errors for two selected data sets. Ideally, the indicator and error

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880



(a) Correlation Error – Indicator

(b) Influence of $|I|$

(c) Influence of γ – Time

(d) Influence of γ – Error

(e) Influence of α – Error

(f) Influence of α – Models

Fig. 8. Parameter Analysis

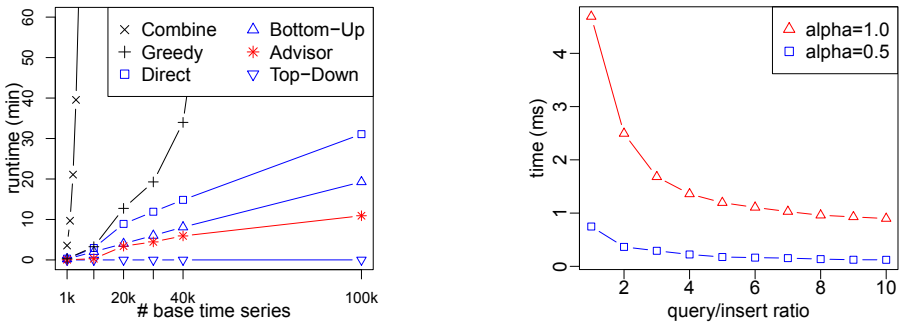
values should be exactly the same and positioned on the straight line in Figure 8(a). For the $Sales$ data set, most of the points are very close to the ideal line showing a good correlation of the indicator with the real forecast error. The $Tourism$ data set exhibits more outliers, but in general shows a good correlation as well. Therefore, our approach to calculate the indicators is valid and heuristically captures the real errors.

Indicator Size: The size of the indicators $|I|$ regulates the number of nodes that are considered in derivation schemes. Figure 8(b) shows the configuration forecast error in relationship with the indicator size $|I|$. For the three real-world data sets, the error decreases with increasing indicator size as more derivation schemes are possible. Our current strategy includes the closest nodes in the indicators if the size $|I|$ is restricted. For that reason the highest error decrease can be observed in the beginning as nearby nodes (exhibiting similarities in some dimensional attributes) are more suitable for forecast derivations. In contrast, the synthetic data set shows only small error improvements with increasing indicator size. For this data set the time series were randomly generated and do not include correlations with respect to the dimensional attributes. The advisor is able to find similar nodes even when a fraction of all available derivations is utilized.

Candidate Selection: The parameter γ affects the number of positive candidates. To analyze the effect of the parameter γ, we use the $Sales$ data set and artificially vary the time that is required to create a single forecast model. We then measured the runtime of the different approaches (Figure 8(c)). As expected, the Greedy, Direct and Top-Down approach increase linearly with the model creation time. The Bottom-Up approach exhibits very similar runtime like the Direct approach and was omitted in this diagram. In contrast, the model configuration advisor only shows a slight increase in runtime. The reason lies in the setting of the parameter γ in the control phase. With increasing model creation time, more models are analyzed in the candidate selection phase in order to reduce the number of models that need to be created in the evaluation phase. The effect of this approach on the forecast error is shown in Figure 8(d). Even as fewer models are created due to high model creation time, the final configuration of the $Sales$ data set leads to the same accuracy, due to the good correlation of the indicators with the real errors. For the other two data set, this correlation is slightly worse, leading to a slight decrease of the forecast error.

Acceptance Criteria: Our goal of adding most beneficial models first to a configuration is regulated by the parameter

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880



(a) Scalability

(b) Forecast Query Runtime

Fig. 9. Runtime Analysis

α. Figure 8(e) shows the development of the configuration forecast error with increasing α. As the advisor choses most beneficial models first, the highest error decrease can be observed in the beginning. For larger αs only small error improvements are visible. For $\alpha = 0.5$, most data sets exhibit a configuration error very close to the best possible error. In Figure 8(f) the corresponding relative number of models are displayed. To achieve the error of $\alpha = 0.5$ less than 15% of the models are necessary. Even for the highest value of α, indicating the best possbile configuration, the percentage of models does not exceed 40%. Thus, with only a fraction of the available models, we can achieve very high accuracy, allowing us to reduce the number of models we need to maintain.

D. Runtime Analysis

In a final series of experiments, we analyzed the runtime of the model configuration advisor. We use the synthetic data set $GenX$ and vary the number of base time series $X \in \{1k, 10k, 20k, 30k, 40k, 100k\}$. We measured the total time that is required to create a configuration for all approaches. For the model configuration advisor, we set α to 0.5, since the previous experiments have shown an already good forecast accuracy with such choice of α. As seen in Figure 9(a), the direct and bottom-up approach increase linear with the data size as the configuration creation consists only of model creation costs, where the bottom-up approach exhibits a lower runtime as only models for base nodes in the time series graph are created. In contrast, the top-down approach shows a constant runtime as it only creates one model for the top node. The runtime of the Greedy approach strongly increases with increasing number of time series as in each iteration the benefit of a model for each node, which does not contain a model, is analyzed. In contrast, our advisor only analyzes the nodes chosen in the candidate selection phase and, thus, shows a better runtime than all other approach, except for the top-down approach. The dominant factor in the runtime of the advisor is given by the creation of the forecast models, which leads to a linear behavior as well. We also denoted the runtime of the Combine approach, which strongly increases with increasing data set size as it requires the computation of a regression matrix over all base forecasts. However, the execution times for the Combine approach were measured in R and are not fully comparable with our implementation in Java.

Finally, we loaded the model configuration into F^{2}DB, using the $Gen10k$ data set for two different parameters of α (0.5 and 1.0). The loading time was less than one minute including storing the time series graph, derivation schemes as well forecast model and weight calculation. We then generated random forecast queries for base and aggregated time series. The average accuracy of these forecast queries confirmed with out measurements in Subsection VI-B. Figure 9(b) shows the average runtime of a forecast query, where we varied the proportion of forecast queries and inserts (representing new time series values) over a period of 10 points in time. We started with one forecast query per insert (query/insert ratio = 1), which resulted in about 150,000 forecast queries and 150,000 inserts. We increased the number queries per insert up to 10 leading to approximately 1,500,000 queries and 150,000 inserts. As visible in Figure 9(b), the average runtime of a single forecast query is very low as models are already precomputed and just maintained if necessary. We can observe a higher runtime for a configuration with $\alpha = 1$ then for $\alpha = 0.5$ as in the second case fewer models are stored. Additionally, the average runtime decreases with increasing number of queries per insert as less maintenance is necessary.

VII. Related Work

The integration of statistical methods, including time series forecasting, into database management systems is getting more and more attention. Recently, Akdere et. al. [26] published a high-level description of a database system that integrates regression and classification on ordinary relations. In our research, we focus only on time series forecasting (requiring different models) and discuss challenges specific for this task. The processing of forecast queries was first introduced within the Fa system [7] that proposed an incremental approach to build models for multi-dimensional time series. Later, a skip-list approach [8] was published for very large time series in order to enable the determination of a suitable history length for model building. All these approaches investigate how to efficiently find the best model for one specific forecast query.

Final edited form was published in "2013 IEEE 29th International Conference on Data Engineering (ICDE). Brisbane 2013", S. 853-864, ISBN 978-1-4673-4908-6
10.1109/ICDE.2013.6544880

We focus on forecasting time series in multi-dimensional settings that allows exploiting models at different dimensions.

The challenge of forecasting high-dimensional data, while maintaining real-time response, was also considered by Agarwal et. al. [5]. They propose to store and forecast only a sub-set of attribute combinations and compute other combinations from those using high-dimensional attribute correlation models. These correlation models are similar to our general derivation scheme introduced in Subsection II-C. However, in their work, the set of models is selected manually for historical importance and seasonality, while we propose an automatic approach to determine the optimal set of models to store. Another work in this area provides a framework for multi-dimensional regression analysis of time-series stream data [27]. Mathematical properties of regression analysis allow the calculation of regression models at higher aggregation levels from a small number of numerical values stored on base levels. More sophisticated models as usually applied in time series forecasting can not exploit such properties and require an empirical selection approach as proposed in this paper.

Determining the best aggregation level for forecasting is related to hierarchical forecasting techniques [23]. In this context, the majority of the literature has focused on comparing bottom-up (models are created for base time series) [10] and top-down (models are created for aggregated time series) [25] approaches. Recently, Hyndman et. al. [17] proposed a general hierarchical forecasting framework that independently forecasts all series at all aggregation levels of the hierarchy and uses a regression model to optimally combine and reconcile these forecasts. However, their approach is only applicable for a small number of time series (as shown in our experimental evaluation) and, in contrast to our work, only maximizes the accuracy and does not take model costs into account.

Finally, optimizing queries in multi-dimensional data sets is related to the computation of multi-dimensional group by queries [28] and materialized view selection [29]. This work is orthogonal to ours and can be used to optimize the calculation of aggregated time series to speed up model maintenance.

VIII. Conclusions

We proposed a model configuration advisor that returns a configuration of forecast models for a given time series data set in multi-dimensional data cubes. By deriving and combining forecasts for a time series from one or multiple forecast models based on other time series, we are able to increase forecast accuracy as well as save model costs. Our general framework is based on an iterative process that selects a set of candidate time series in each iteration for which a model should be built and analyzed. Parameters of the advisor like the number of candidate models are automatically tuned in a control phase. In our evaluations we show that our advisor is able to achieve a higher forecast accuracy than other approaches and that we are able to calculate a configuration in reasonable time even for larger data sets and complex models. Finally, we are able to load a model configuration into our existing PostgreSQL prototype F^{2}DB and efficiently process forecast queries.

References

[1] 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.
[2] J. T. Mentzer and C. C. Bienstock, "The seven principles of sales-forecasting systems," *Supply Chain Management Review*, 1998.
[3] H. Feng, N. Lumineau, M. Hacid, and R. Domps, "Hierarchy-based update propagation in decision support systems," in *DASFAA*, 2012.
[4] 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.
[5] D. Agarwal, D. Chen, L. Lin, J. Shanmugasundaram, and E. Vee, "Forecasting high-dimensional data," in *SIGMOD*, 2010.
[6] Y. Cui, R. Zhang, W. Li, and J. Mao, "Bid landscape forecasting in online ad exchange marketplace," in *KDD*, 2011.
[7] S. Duan and S. Babu, "Processing forecasting queries," in *VLDB*, 2007.
[8] T. Ge and S. Zdonik, "A skip-list approach for efficiently processing forecasting queries," in *VLDB*, 2008.
[9] G. Fliedner, "An investigation of aggregate variable time series forecast strategies with specific subaggregate time series statistical correlation," *Computers & Operations Research*, vol. 26, pp. 1133–1149, 1999.
[10] D. Dunn, W. Williams, and T. DeChaine, "Aggregate versus subaggregate models in local area forecasting," *Journal of the American Statistical Association*, vol. 71, pp. 68–71, 1976.
[11] G. Box, G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*. Wiley, 2008.
[12] U. Fischer, F. Rosenthal, and W. Lehner, "F2db: The flash-forward database system," in *ICDE*, 2012.
[13] V. Harinarayan, A. Rajaraman, and J. D. Ullman, "Implementing data cubes efficiently," in *SIGMOD*, 1996.
[14] J. G. D. Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International Journal of Forecasting*, vol. 22, pp. 443–473, 2006.
[15] S. Makridakis and M. Hibon, "The M3-Competition: results, conclusions and implications," *International Journal of Forecasting*, vol. 16, pp. 451 – 476, 2000.
[16] C. W. Gross and J. E. Sohl, "Disaggregation methods to expedite product line forecasting," *Journal of Forecasting*, vol. 9, pp. 233–254, 1990.
[17] R. J. Hyndman, R. A. Ahmed, G. Athanasopoulos, and H. L. Shang, "Optimal combination forecasts for hierarchical time series," *Computational Statistics and Data Analysis*, 2011.
[18] R. J. Hyndman and A. B. Koehler, "Another look at measures of forecast accuracy," *International Journal of Forecasting*, vol. 22, pp. 679 – 688, 2006.
[19] U. Fischer, M. Böhm, and W. Lehner, "Offline design tuning for hierarchies of forecast models," in *BTW*, 2011.
[20] G. Athanasopoulos, R. A. Ahmed, and R. J. Hyndman, "Hierarchical forecasts for australian domestic tourism," *International Journal of Forecasting*, vol. 25, pp. 146–166, 2009.
[21] (2012) Tourism Research Australia - National Visitor Survey. [Online]. Available: http://www.ret.gov.au/tourism/research/tra/Pages/default.aspx
[22] (2012) The MeRegio Project. [Online]. Available: http://www.meregio.de/en/
[23] G. Fliedner, "Hierarichal forecasting issues and use guidelines," *Industrial Management & Data Systems*, vol. 101, pp. 5–12, 2001.
[24] A. Zellner and J. Tobias, "A note on aggregation, disaggregation and forecasting performance," *Journal of Forecasting*, vol. 19, pp. 457–469, 2000.
[25] E. B. Fliedner and V. A. Mabert, "Constrained forecasting: Some implementation guidelines," *Decision Sciences*, vol. 23, pp. 1143–1161, 1992.
[26] M. Akdere, U. etintemel, M. Riondato, E. Upfal, and S. Zdonik, "The case for predictive database systems: Opportunities and challenges," in *CIDR*, 2011.
[27] Y. Chen, G. Dong, J. Han, B. W. Wah, and J. Wang, "Multi-dimensional regression analysis of time-series data streams," in *VLDB*, 2002.
[28] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, and S. Sarawagi, "On the computation of multidimensional aggregates," in *VLDB*, 1996.
[29] S. Agrawal, S. Chaudhuri, and V. R. Narasayya, "Automated selection of materialized views and indexes in sql databases," in *VLDB*, 2000.