Ulrike Fischer, Frank Rosenthal, Wolfgang Lehner

F²DB: The Flash-Forward Database System

Erstveröffentlichung in / First published in:

2012 IEEE 28th International Conference on Data Engineering. Arlington, 01.-05.04.2012. IEEE, S. 1245-1248. ISBN 978-0-7695-4747-3.

DOI: http://dx.doi.org/10.1109/ICDE.2012.117

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1245-1248, ISBN 978-0-7695-4747-3
https://doi.org/10.1109/ICDE.2012.117

F²DB: The Flash-Forward Database System

Ulrike Fischer, Frank Rosenthal, Wolfgang Lehner

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

***Abstract*—Forecasts are important to decision-making and risk assessment in many domains. Since current database systems do not provide integrated support for forecasting, it is usually done outside the database system by specially trained experts using *forecast models*. However, integrating model-based forecasting as a first-class citizen inside a DBMS speeds up the forecasting process by avoiding exporting the data and by applying database-related optimizations like reusing created forecast models. It especially allows subsequent processing of forecast results inside the database. In this demo, we present our prototype F²DB based on PostgreSQL, which allows for transparent processing of *forecast queries*. Our system automatically takes care of model maintenance when the underlying dataset changes. In addition, we offer optimizations to save maintenance costs and increase accuracy by using derivation schemes for multidimensional data. Our approach reduces the required expert knowledge by enabling arbitrary users to apply forecasting in a declarative way.**

I. Introduction

Modern data analysis in data-warehouse systems involves increasingly sophisticated statistical methods that go well beyond the rollup and drilldown over simple aggregates of traditional BI [1]. As data warehouse systems implicitly guarantee to contain a time dimension, stored data usually constitutes time series (e.g., sales per month). This data is used as basis of decision-making (e.g., planning of production). Such decisions benefit from having *forecasts* of important metrics (e.g., expected demand or inventory).

Many existing commercial database systems offer some kind of regression modeling. Although general regression may be used to extrapolate overall trends, specialized time series models offer a higher prediction accuracy. Specific methods for forecasting often involve the specification of a stochastic model that captures some form of *auto-regression*. We will refer to such models as *forecast models*. Specific methods for forecasting are offered only in a minority of common database systems in a limited way (Section IV).

Therefore, a common workflow for forecasting consists of exporting the data to statistical tools, like Matlab or R, and choosing and applying forecast methods externally. However, pushing computation directly to the data has several advantages. First, the knowledge encoded in models by domain experts can be stored and reused. This allows other users, especially those not trained in forecasting, to benefit from forecasts based on these models. Second, the forecast itself can be used inside the DBMS and processed with other source data (e.g., through joins). Third, the separation of conceptional and physical layer in a DBMS opens a wide variety of optimization potential. For example, we do not need to store a physical model for every single logical forecast query. Instead, by using transparent derivation schemes, we can reduce the number of models that need to be created, stored and maintained.

We propose a solution that integrates forecasting natively into an existing DBMS. In contrast to flash back queries that allow a view on the data in the past (`AS OF`), we have developed a *Flash-Foreward Database System* (F²DB). It offers a simple interface to make forecasting usable for any database user and hides complex internals like automatic maintenance of forecast models. For example, this simple forecast query returns estimates of sold quantities for mobile phones in the state Ohio for the next month:

```
SELECT   orderdate, SUM(sales)
FROM     facts
WHERE    product='phones' AND state='Ohio'
GROUP BY orderdate
AS OF    current_date + interval '1 month'
```

In summary, we make the following contributions: (1) We provide an integrated language to define forecast models and to submit declarative forecast queries. For this, we introduce new operators that are compatible to existing relational operators. (2) We provide a system that transparently processes forecast queries and automatically maintains forecast models when base data evolves. (3) We introduce an additional model advisor that aims to optimize forecast queries (speed and accuracy) by suggesting derivation schemes.

II. System Overview

A general overview of our forecasting system is shown in Figure 1. Models of time series are represented as database objects that are managed in a *model pool* (center of Figure 1). The *model index* makes models quickly accessible for model usage and is necessary for identifying models that are affected by updates to base data [2]. F²DB provides language constructs to create new models and add them to the pool (left in Figure 1). This can be done manually, which is a way for experts to store their knowledge in F²DB, or by using algorithms that automatically select the "best" model for a time series, e.g., based on the *Akaike Information Criterion* [3]. After a model is indexed in the model index, it can be automatically

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1245-1248, ISBN 978-0-7695-4747-3
https://doi.org/10.1109/ICDE.2012.117

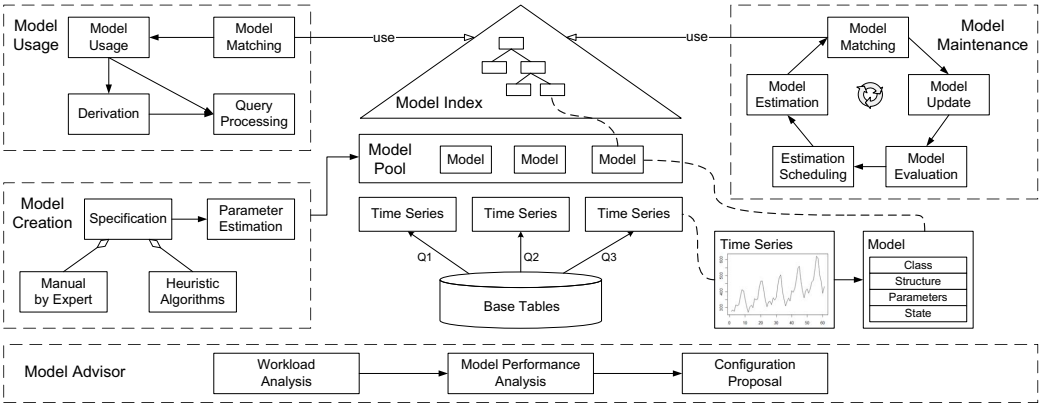

Fig. 1. System Overview

and transparently used by forecast queries. *Model matching* is necessary to identify the concerning model(s). Matching may also return models of related time series that can be used to derive the requested forecast (Section III-B). After usage and possible derivation, the forecast can be further processed using regular relational primitives.

Models need to be maintained when new data arrives (right in Figure 1). First, a matching step is required to find the affected model(s) using the model index. Next, the model state, i.e., the internal history, is updated. However, updates may also reflect changes in the time series behavior and these can only be incorporated by reestimating model parameters. Since this step is computationally expensive we propose several possibilities to reduce the effort for maintenance. First, *model evaluation* checks the need for maintenance (e.g., through an error threshold). Second, we reduce maintenance effort through *estimation scheduling* (Section III-C).

Last, we provide a *model advisor* [4] component that analyzes past query workloads and model performance (cost and accuracy). Consequently, it might propose a different configuration of models in the model pool (Section III-B).

III. System Details

Forecasting has been studied intensively in various domains, which leads to a very large number of possible forecast methods. For example, the machine learning domain offers approaches that involve the use of support vector machines [5] or extended decision trees [6], while the field of statistics yields classes of models like exponential smoothing (EXP) models [7] or the autoregressive integrated moving average (ARIMA) models [8].

From a usage perspective, these diverse methods all share a simplified interaction paradigm, where (1) model parameters are estimated to fit a specified model to a time series and (2) the model is applied to forecast future data.

F^{2}DB provides a general architecture that allows the integration of any forecast method. We use the extension capabilities of PostgreSQL to provide a generic interface where a new forecast method is added by implementing predefined functions and registering them in the system catalog. However, for the purpose of this demo we selected two representatives, exponential smoothing and ARIMA models. Both are well examined [9], have shown empirically to be able to model a wide range of real world time series and are usually computationally more efficient than elaborate machine learning approaches.

A. Model Specification and Representation

The creation of a model can be triggered using a `CREATE MODEL` statement that specifies which model to fit to which time series. The time series is defined by a regular `SQL` query. The only constraints to this subquery are naturally that there needs to be one value attribute and at least one ordering attribute. For example, in order to create a forecast model for the forecast query shown in the introduction of this paper, we provide the following `SQL` statement:

```
CREATE MODEL m1
FOR FORECAST OF sales ON orderdate
ALGORITHM arima AUTOMATIC SEARCH
TRAINING_DATA
  SELECT   orderdate, SUM(sales)
  FROM     facts
  WHERE    product='phones' AND state='Ohio'
  GROUP BY orderdate
```

The model is then stored in the model pool. This involves storing the model class, model structure, model parameters and current model state. For fast retrieval of forecast models, we internally provide a special index structure over all models [2]. In order to enable specific derivation schemes (Section III-B) we explicitly encode functional dependencies in this index. To exploit existing functional dependencies, we provide a `CREATE FORECAST HIERARCHY` statement. Our example forecast query results from a hierarchy of store locations that leads to the model index shown in Figure 2. Sales for stores in cities (Sacramento, Parma, Toledo) can be aggregated to sales over whole states (California, Ohio) and finally to total phone sales over all locations. Models are always stored in the leaves of this index. The node *[Total]* below the node *Ohio* represents sales over all cities in the state Ohio and therefore references the model *m1* created before.

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1245-1248, ISBN 978-0-7695-4747-3
https://doi.org/10.1109/ICDE.2012.117

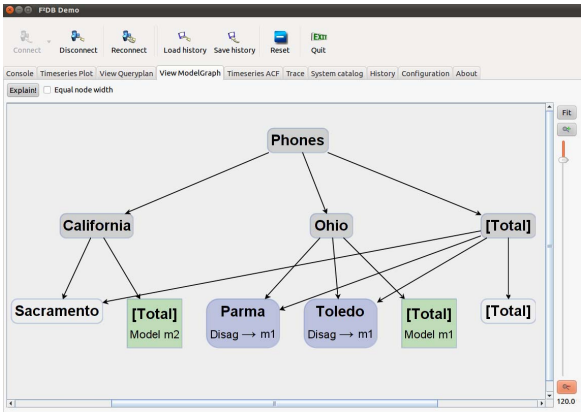

Fig. 2. Example Model Index

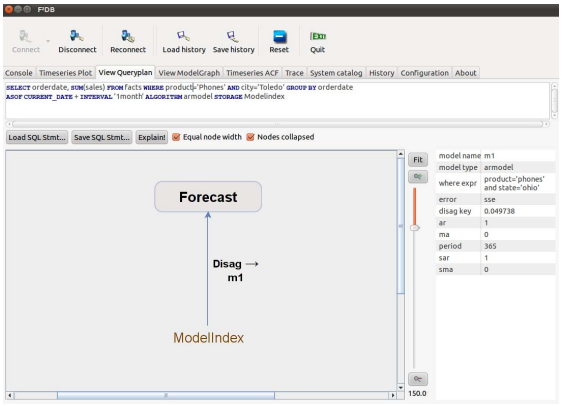

Fig. 3. Example Execution Plan

B. Model Usage with Derivation Schemes

A forecast query declaratively defines a time series that our system should forecast. When such a query is issued we need to find out whether there are suitable models in the model pool. If no model is found in this process an error is returned to the user. Otherwise, there are three different ways to answer the forecast query: direct, aggregation or disaggregation.

Direct In the simplest case, model usage returns the forecasts of the model that fits exactly the queried time series.

Aggregation However, as we store hierarchical dependencies in the model index, we can use it to lookup whether we can derive forecast values of a query by aggregation of forecast values at a lower hierarchy level. For example, a query that requests forecasts over all phones in all locations can be answered by using the forecast models $m1$ and $m2$ for sales in Ohio and California (Figure 2). The individual forecast values of these models are aggregated to get the final result.

Disaggregation Finally, we can use the model index to find a model at a higher level of the aggregation hierarchy and use disaggregation to derive the forecast on the target level. However, this requires a disaggregation scheme that specifies how to scale down the high level forecast to retrieve low level forecast values (e.g., the historical fraction). Therefore, we provide the possibility to explicitly create a disaggregation scheme in the model index by using the `CREATE DISAG SCHEME` command. The example model index in Figure 2 contains two disaggregation schemes, for sales in Parma and Toledo, which both reference the model $m1$. Assume, a user wants to forecast sales for phones in Toledo for the next month. The execution plan of this query (Figure 3) consists of only one forecast operator that gets as input a disaggregation scheme and a reference to the model $m1$. Internally the model $m1$ is used to create forecast values for sales in Ohio and multiplied with a disaggregation key to produce the final output. To answer this query no access to the base data is necessary as all required information is stored in the model index. The appropriate derivation scheme is chosen transparently.

Model Advisor In contrast to materialized view selection, a derivation scheme influences the accuracy of the output compared to the accuracy of a model specifically created for a given query [10]. Some derivation schemes might even increase the accuracy. For example, the time series over sales in Toledo might contain a lot of random fluctuations but the general trend is similar to other cities in Ohio. Therefore, a more robust forecast may be created by using the model for sales in Ohio and by applying a disaggregation scheme. In order to exploit the potential of derivation schemes, we provide an additional model advisor component [4]. Our model advisor takes as input current models in the system, a history of forecast errors over these models and a workload consisting of forecast queries and updates. Using this input, it either proposes to drop models and to create disaggregation schemes in order to save maintenance cost or to create new models in order to increase forecast accuracy.

C. Model Maintenance

The goal of model maintenance is to provide a logical consistency between a time series and the models that are based on it. First, maintenance involves the update of the model state and disaggregation scheme. However, the most expensive part in model maintenance is parameter reestimation, since estimators for typical time series models are implemented using numerical procedures and involve optimization algorithms that evaluate complex cost functions many times over all available data of the time series. To address this issue, we first evaluate models by using the updates of the time series to determine the prediction error and trigger estimation only when necessary. To determine the best point of reestimation, we provide simple strategies (e.g., time-based or by using an error threshold) as well as more sophisticated ones that assess the need for parameter reestimation based on a synopsis that can be maintained incrementally [11]. Second, we perform maintenance asynchronously. We therefore delay maintenance to whenever the system has low CPU utilization, so writers do not have to pay for model maintenance. However, if a query references a model that is not up to date, we trigger immediate parameter reestimation if required. Only in this case the reader has to pay for estimation effort in terms of longer

Final edited form was published in "2012 IEEE 28th International Conference on Data Engineering. Arlington 2012", S. 1245-1248, ISBN 978-0-7695-4747-3
https://doi.org/10.1109/ICDE.2012.117

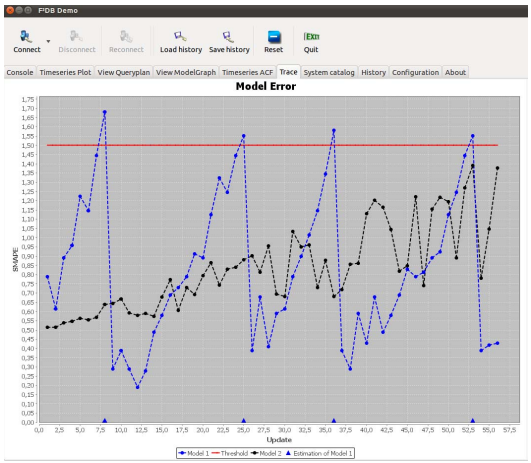

Fig. 4. Example Maintenance Trace

query response times.

IV. Related Work

Forecasting is supported as extension of only two common database systems. Oracle offers a `FORECAST` command as part of its OLAP DML [12]. Therefore, it cannot be used as part of regular SQL queries. The Data Mining Extension (DMX) in Microsoft SQL Server supports forecasting of time series using a custom forecast algorithm, which offers the interface method `PredictTimeSeries` [13]. DMX allows to query models itself for predictions, but lacks the transparency of our seamless integration that does not require the selection of a model in a forecast query. In addition, both products do not provide automatic maintenance of models. Recently, Akdere et. al. [14] published a high-level description of a database system that integrates regression and classification on ordinary relations. In our work, we focus on time series forecasting (requiring different models) and employ techniques specific for this task like model reuse with derivation schemes as well as time series model maintenance.

V. Details of the Demo

Our prototype F^{2}DB is based on PostgreSQL and integrates the different components described in this paper. Our demonstration contains different workloads and scenarios to show how F^{2}DB works and how the system scales.

Workloads We have obtained several real-world datasets, which we use with different forecasting workloads in our system. This includes hierarchical data sets to demonstrate model usage with derivation schemes and the benefit of our model advisor. For example, one dataset was obtained from the Tourism Research Australia (http://www.ret.gov.au/tourism/tra/domestic/national/Pages/default.aspx) and consists of observations on the number tourists in Australia (according to state and purpose of visit). In addition, we will provide real-world datasets from the energy domain. These datasets pose the challenge of high update intervals and demonstrate the necessity of our maintenance scheme.

Scenarios Our demonstration focuses on three different aspects: (1) user view, (2) global view and (3) maintenance.

First, a user can create forecast models manually using `SQL` (see the Example in Subsection (III-A)) or by a provided graphical interface. The user can submit declarative forecast queries and view the resulting query plan (Figure 3). The user can also view a graphical representation of the forecast results as well as additional information like confidence intervals in order to gain insight into the concepts and accuracy of the different derivation schemes.

From a global point of view, a user can view and manipulate the model pool by using our model index (Figure 2). Different workloads can be submitted to our system and statistics like the maintenance costs, forecast accuracy or query throughput can be viewed. This is also an interactive part where the demo visitor can change workloads (e.g., query and insert frequency) and parameters (e.g., model types, maintenance strategy). In addition, we prepared some demo runs in order to benchmark the performance of different model configurations (e.g., all possible models vs. advisor recommendation).

Last, we will demonstrate in detail different strategies to maintain models and to determine the need for parameter reestimation. For example, Figure 4 shows a screenshot of short error traces of two models over time. We see the result of one strategy where maintenance is triggered when an error threshold is exceeded (red line). In this example, estimation is triggered four times for one model (blue), while the other model caused no maintenance cost whatsoever (black). After every maintenance step, the forecast error is reduced.

References

[1] J. Cohen, B. Dolan, M. Dunlap, J. M. Hellerstein, and C. Welton, "Mad skills: New analysis practices for big data," in *VLDB*, 2009.
[2] U. Fischer, F. Rosenthal, M. Boehm, and W. Lehner, "Indexing forecast models for matching and maintenance," in *IDEAS*, 2010.
[3] R. J. Hyndman and Y. Khandakar, "Automatic time series forecasting: The forecast package for r," *Journal of Statistical Software*, vol. 27, pp. 1–22, 2008.
[4] U. Fischer, M. Boehm, and W. Lehner, "Offline design tuning for hierarchies of forecast models," in *BTW*, 2011.
[5] K.-R. Müller, A. J. Smola., G. Rätsch, B. Schölkopf, J. Kohlmorgen, and V. Vapnik, "Predicting time series with support vector machines," in *ICANN*, 1997.
[6] C. Meek, D. M. Chickering, and D. Heckerman, "Autoregressive tree models for time-series analysis," in *SIAM*, 2002.
[7] R. J. Hyndman, A. B. Koehler, R. D. Snyder, and S. Grose, "A state space framework for automatic forecasting using exponential smoothing methods," *International Journal of Forecasting*, vol. 18, 2000.
[8] G. B., G. Jenkins, and G. Reinsel, *Time Series Analysis: Forecasting and Control*. Wiley, 2008.
[9] J. G. D. Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International Journal of Forecasting*, vol. 22, pp. 443–473, 2006.
[10] G. Fliedner, "Hierarichal forecasting issues and use guidelines," *Industrial Management & Data Systems*, vol. 101, pp. 5–12, 2001.
[11] F. Rosenthal and W. Lehner, "Effcient in-database maintenance of arima models," in *SSDBM*, 2011.
[12] Oracle, "Oracle OLAP DML Reference: FORECAST - DML Statement," 2011.
[13] PredictTimeSeries – Microsoft SQL Server 2008 Books Online, "http://msdn.microsoft.com/en-us/library/ms132167.aspx," 2011.
[14] M. Akdere, U. etintemel, M. Riondato, E. Upfal, and S. Zdonik, "The case for predictive database systems: Opportunities and challenges," in *CIDR*, 2011.