Forecasting in Database Systems

Ulrike Fischer

Database Technology Group
Technische Universität Dresden
Nöthnitzer Str. 46
01187 Dresden
ulrike.fischer@tu-dresden.de

Abstract: Time series forecasting is crucial in a number of domains such as production planning and energy load balancing. In these areas, forecasts are often required by non-expert users on large multi-dimensional data sets expecting short response times. However, as current traditional database systems support forecasting only in a limited and non-declarative way, it is performed outside the database system by specially trained experts. We introduce a novel approach that seamlessly integrates time series forecasting into an existing database management system. In contrast to flash-back queries that allow a view on the data in the past, we have developed a Flash-Forward Database System (F^{2}DB) that provides a view on the data in the future. It supports a new query type — a forecast query — that enables forecasting of time series data for any user and is automatically processed by the core engine of an existing DBMS. We introduce various optimization techniques for three different types of forecast queries: ad-hoc queries, recurring queries, and continuous queries. All approaches intend to increase the efficiency of forecast queries while ensuring high forecast accuracy.

1 Introduction

Time series forecasting has been subject to intensive research for a long period of time and still offers many open research problems [GH06]. This comes as no surprise as forecasting is crucial for a number of application areas. The Data Warehousing Institute (TDWI) surveyed about 400 IT professionals and data analysts, where nearly 80% of the respondents said "reacting more quickly to changing market conditions is a business benefit they seek" [Sto12]. The overall business goals are often to use predictive insights to anticipate, rather than react to customer behavior.

In the past, time series forecasting was performed by highly qualified statistical experts, with long experience in the company, who manually experimented with different forecasting algorithms and parametrizations using dedicated statistical software environments. Over the last few years, we are experiencing a paradigm shift, which is not just related to forecasting but valid for any kind of sophisticated statistical algorithm. We can observe the transition of traditional database systems to big data stores where massive amount of data arrives continuously in real-time from vast data sources. This development results not only in big challenges but also big opportunities for analyzing large data sets in real-time, which further leads to better-informed business decisions. As a consequence, modern data analysis in database management systems involves increasingly sophisticated statistical methods that go well beyond the roll-up and drill-down over simple aggregates of traditional BI [CDD$^+$09]. Moreover, sophisticated models and advances in autonomic model selection and self-management make advanced data analytics more useful for users and applications. On the one hand, data mining is increasingly performed by people who are not mathematicians or statistical experts. Many users do not want to concern themselves with tasks like model selection and training and are satisfied with "good enough" forecasts [ACR$^+$11]. On the other hand, there is a frequent need for fully automatic forecasting without any human intervention [HKSG02], e.g., balancing energy demand and supply.

Several research groups (e.g., [CDD$^+$09], [KNR13]) and major database vendors (e.g., [MYC05], [GLW$^+$11]) are making an effort towards integrating sophisticated statistical algorithms, such as time series forecasting, into classical data management platforms. However, many existing approaches follow a black-box style and try to keep changes to the database system as minimal as possible, for example, by using user-defined functions or by connecting the database system with a statistical software environment. While such approaches are more general and easier to realize, they miss significant opportunities for performance improvements. Moreover, most methods still approach the problem with statistical models and statistical algorithms, and try to get a database system to act like a statistical software environment. Such approaches ignore the concept of declarative database queries that do not care about the actual computation of query results.

In the thesis [Fis14], we introduce a novel approach that seamlessly integrates time series forecasting into an existing database management system. A tight coupling of the time series forecasting process with a DBMS ensures consistency between data and models and increases the usability as well as the efficiency of the forecasting process. In contrast to flash back queries that allow a view on the data in the past, we have developed a *Flash-Forward Database System* (F^{2}DB) that provides a view on the data in the future. It supports a new query type — a forecast query — that enables forecasting of time series data and is automatically and natively processed by the core engine of an existing DBMS.

In the following brief summary of the thesis, Section 2 gives a background on time series forecasting. Section 3 introduces the conceptional architecture of our flash-forward database system. Sections 4–6 highlight the different components and optimization techniques of F^{2}DB. Finally, we conclude in Section 7.

2 Foundations of Time Series Forecasting

Time series forecasting is best described by a generalized model-based forecasting process, from which we can derive typical requirements on a forecasting system as well as realization possibilities inside a DBMS.

Model-Based Forecasting A *time series* is a sequence of observations taken sequentially in time, spaced at equidistant time intervals. *Forecasting* refers to the estimation of values of a time series at future points in time, so called *forecast values* or short *forecasts*.

Hereby, the *forecast horizon* describes the interval from the next point in time up to a given future point in time. A *forecasting method* is a procedure for computing forecasts from present and past values. Most forecasting methods are based on a *forecast model*, which is learned over historical training data and used to compute forecast values (e.g., Exponential Smoothing or ARIMA models [BJR08]). *Model-based time series forecasting* consists of three essential steps. First, a forecast model is created by defining input, output as well as the forecasting method, and by estimating the model parameters (*model creation*). Second, forecast values based on the created forecast model are calculated (*model usage*). Last, the forecast model is evaluated by comparing real time series data with forecast values and, optionally, model adaption is triggered by recalculating the model parameters or choosing a new model (*model maintenance*). In model-based forecasting, the time-consuming part poses the model creation step, whereas model usage only requires the application of a function with the trained parameters.

System Requirements Time series forecasting is an important technique for various application areas such as production planning [MB98], energy balancing, and online display advertisement. A system that supports the introduced model-based forecasting process as well as typical application domains has to follow certain requirements. First, we require a suite of widely used forecasting techniques applicable for different areas. Furthermore, a generic interface would be desirable that enables the inclusion of new domain-specific forecasting methods. A simple and declarative query language allows the application of forecasting by any users without statistical expertise (e.g., supply chain managers). Real-time requirements (e.g, a few hundred milliseconds in display advertisement) over large multi-dimensional data sets demand the efficient creation of forecast models. However, as model creation is expensive, mechanisms have to be provided that allow the storage and reuse of forecast models for different queries and users. Consequently, forecast models have to be continuously and efficiently adapted to changes in the time series behavior.

Architectural Integration Numerous works have addressed the integration of analytical methods inside a DBMS. Existing methods can be classified into (1) no-database integration approaches that use external software (Matlab, R), (2) partial integration approaches that try to keep changes to the database as small as possible (e.g., [CDD$^+$09, GLW$^+$11]), and (3) full integration approaches that actually extend the functionality of a database system (e.g., [DM06, ACR$^+$11]). External systems often contain a large number of statistical forecasting methods, but lack database features such as declarative forecast queries, query optimization, or model storage and reuse. Partial integration approaches take a first step towards the integration of the forecasting lifecycle within a DBMS and integrate forecasting into relational query processing to some extent. They, however, still treat the forecasting process as a black-box approach, either within a user-defined, a customized function, or within a R script. Subsequently, all decisions have to be made locally within the specific function, allowing no joint model reuse, no automatic maintenance, or multi-dimensional physical design. In contrast, models are handled as first class citizens and automatically maintained by some of the full integration approaches, such as MauveDB [DM06]. However, existing approaches do not support declarative forecast queries as they require the explicit selection of a model in a query. Furthermore, most approaches do not focus on time series forecasting techniques and optimizations.

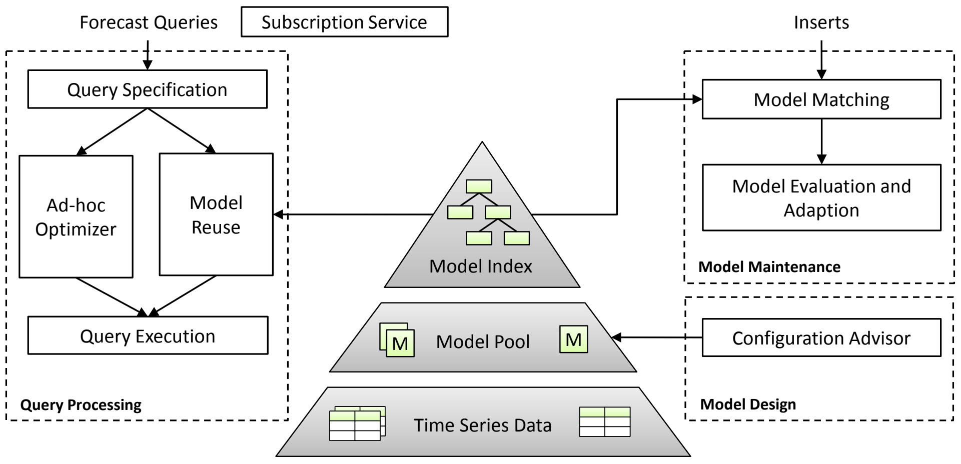


Figure 1: Flash-Forward Database System F^{2}DB

3 A Flash-Forward Database System

Based on the discussed system requirements and the limitations of existing work, in this section, we propose a general architecture that integrates the whole forecasting lifecycle natively into a database management system. In contrast to flash-back queries that allow a view on the data in the past, we propose a *Flash-Foreward Database System* (F^{2}DB) that provides a view on the data in the future. Figure 1 shows the main conceptional components of F^{2}DB . Based on this architecture, a prototype was developed within the open-source DBMS PostgreSQL [FRL12a].

Analogue to traditional database views, a *time series* can be specified by arbitrary database queries and represented as a special time series view. *Forecast models* on time series are stored as first-class citizens in a common *model pool*. Furthermore, models are indexed in a specialized index structure that efficiently finds existing models for a given forecast query and insert operation (*model index*).

The processing of declarative forecast queries requires the extension of the traditional query processing engine. *Query specification* recognizes new forecast-specific keywords and further analyzes the given query. Hereby, we can either search and *reuse* existing models from the model index or create new models at query runtime. The latter is supported by the *ad-hoc optimizer*, which includes various optimization techniques to increase query runtime as well as query accuracy. Finally, the query is *executed* by new forecast-specific physical operators. Insert operations, containing new time series tuples, require the maintenance of models in the model pool. Hereby, *model matching* finds existing models that are based on those inserts. Models are then *evaluated* and *adapted* if necessary. Forecast model maintenance poses additional overhead to the database system. In the style of traditional materialized view and index advisors, the *configuration advisor* suggests a physical design of forecast models for a given query workload. Hereby, the advisor is faced with the tradeoff of reducing model maintenance costs and increasing forecast query efficiency and accuracy. A last component, the *subscription service*, allows applications to register a query once at F^{2}DB. As new time series values arrive, it automatically sends notifications to the application based on time and accuracy constraints.

4 Processing Forecast Queries

In this section, we discuss the basics of forecast query processing and propose optimization techniques for ad-hoc forecast queries.

Forecast Queries in SQL Support for forecasting requires the extension of the SQL data manipulation language with forecast-specific keywords. In the SQL:2011 standard, a flashback query retrieves data as it existed at some point in the past using the `AS OF` clause. We use the same language construct to retrieve data as it will exist in the future. For example, the following forecast query requests sales forecasts of mobile phones:

```
SELECT    orderdate, quantity
FROM      facts
WHERE     pgroup = 'phones'
GROUP BY  orderdate
AS OF     now() + interval '3 months'
```

Our goal is to keep the query language as minimal as possible, for which reason only the `AS OF` keyword is mandatory. However, we offer additional language constructs to specify information about input and output variables, or the forecasting method [FRL12a].

Logical Forecast Operator Forecasting introduces a novel semantic that is not covered by existing database operators. Based on existing input tuples, representing a historical time series, new approximate tuples are outputted, representing the future. This leads to a novel logical operator, the *forecast operator*, and interesting interaction schemes with existing operators. Most importantly, as the output of the forecast operator is only approximate, we have to extend the traditional semantic of equivalence of query plans. In the thesis, we analyze the relationship of the forecast operator with traditional database operators and explore possibilities to create a forecast model within a query plan.

Execution Model and Physical Plan Operators Analogue to traditional operators, the forecast operator can be mapped to different physical implementations, covering the variety of available forecasting methods, including general techniques as well as special forecasting methods for different domains. Forecasting methods are captured by a black-box-style generic internal *forecasting method interface* (FMI) that allows the addition of new methods by implementing predefined functions and registering them in the system catalog. Such a black-box approach is not restricted to a particular class of forecasting methods and allows the reuse of existing forecasting libraries behind the simple interface. Following the general forecasting process, the interface decouples model creation and model usage functionality and is exploited by two new physical plan operators. The `CreateModel` operator is responsible for model creation. It receives a time series relation as input and outputs a set of forecast models. Subsequently, the `Forecast` operator receives as input a set of forecast models and outputs a time series relation containing corresponding forecast values. To include the new operators into query optimization, we have developed novel cost models.

Sample-Based Query Optimization In contrast to traditional query optimization, forecast queries are approximate per se. We can reduce the amount of input data to the forecast operator and still compute valid results with a similar accuracy. Sampling based methods reduce the amount of processed data and have been successfully applied in many scenarios. In our case, we apply sampling in two different manners. *Vertical sampling* reduces the number of processed time series in a query [FRL12b], whereas *horizontal sampling* reduces the length of the time series. Both approaches reduce the processed data and subsequently the query runtime without or little loss in forecast accuracy. Vertical sampling results in missing time series in a query, which have to be estimated and integrated into the overall query result. Hence, we need to focus on how to estimate these missing time series. In contrast to traditional sampling that deals with single values, we sample complete time series. We propose novel estimators and sample selection strategies that exploit this fact based on the time series history. Horizontal sampling only reduces the history length, but does not remove intermediate time series values. Subsequently, the forecast value computation is exactly the same, but might result in different accuracy. However, in contrast to vertical sampling, an increasing sample size does not necessarily increase the forecast accuracy. A longer history length might have no effect at all (depending on the forecasting method) or even lead to an accuracy decrease. We introduce an empirical optimization technique that evaluates several history lengths at runtime on a sample of time series.

Evaluation In our evaluation, we compare the end-to-end performance of our in-DBMS implementation to alternative integration approaches and evaluate the overall benefit of the proposed optimization techniques. For our end-to-end comparison, we adapt the classical TPCH benchmark to be able to process forecast queries, denoted as *F-TPCH*. As the TPCH benchmark is centered around a data-warehouse scenario with OLAP-style queries, it is well suited for advanced analytics such as time series forecasting. The adaption of the TPCH benchmark requires changes to the data generation as well as query generation components of the benchmark. Furthermore, to show the accuracy of our techniques, we acquired several real-world data sets from various domains. Our experimental evaluation shows that the integration of forecasting inside the database leads to significant speed ups compared to external forecasting. Query runtime and forecasting accuracy is further improved by applying sample-based optimization.

5 Materialization of Forecast Models

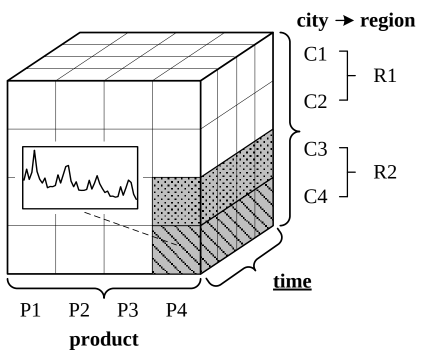
We now turn our attention from ad-hoc to recurring forecast queries. Specifically, our goal is to decrease the runtime and increase the accuracy of forecast queries by preselecting and materializing forecast models. Hereby, we want to retain the transparent processing of forecast queries from the last section, which requires a mechanism to transparently find existing models in the database.

Model Index Materialization of forecast models is realized by two new data structures. The *model pool* contains all currently existing forecast models. Each model is associated with static and dynamic meta data. A second data structure, the *model index*, identifies matching models for incoming query and insert transactions [FRBL10]. The index is kept as in-memory structure, as it needs to be accessed for every forecast query and insert transaction. The key of the model index is the input data of the model, whereas the model itself is the value. Hereby, a predicate index is responsible for finding predicates that satisfy certain query conditions or tuple values, which are further matched against signatures of *where expressions* to actually identify the affected models. This allows arbitrary forecast queries and enables expression sharing of different queries.

Forecast query processing can now take two different routes. On the one hand, if a model is available, forecast values are directly computed from this model without accessing the base data on disk. On the other hand, if no model is found in the model index, we create a new model as done in the previous section. In addition, we store the model in the model index so it can be reused by subsequent queries. As new time series values arrive, we need to identify and maintain all affected models in the model index. First of all, this involves the update of the internal model state to the current time series values, which is a fast and incremental operation. However, updates may also reflect changes in the time series behavior and these can only be incorporated by reestimating the model parameters. Since this step is computationally expensive, we propose several evaluation and adaption strategies to reduce the effort for full maintenance.

Optimization of Model Configurations Forecast model maintenance poses additional overhead to the database system. Materializing and maintaining a model for each single time series is infeasible for large multi-dimensional data sets as found in typical data warehouse environments. Existing forecasting literature already provides some initial techniques to reuse models, for example, by using low-level models to compute forecasts for aggregated time series data. Interestingly, those techniques do not only reduce the number of models in a physical design, they can even increase the accuracy of the forecast result.

Consider the example of forecasting sales data with sales quantity as measure and time, product, city, and region as dimensional attributes (Figure 2). The dimension time together with the measure forms time series for different products and locations. Forecast Query 1 in Figure 2 demands the future of the *base time series* representing the dashed area, i.e., sales forecasts of product $P4$ in city $C4$ over the next day. Forecast Query 2 (dashed 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*. To answer Forecast Query 2 of Figure 2 we may exploit various approaches. Most current state of the art would create a forecast model over the aggregated time series (dashed and dotted area). However, alternatively, we might create two forecast models, one over city $C3$ and one over city $C4$, and calculate the forecast of Query 2 by aggregating the individual forecasts. Third, we could create a model over all cities ($C1$-$C4$) and then use disaggregation to just compute the forecasts of the cities in region $R2$. Finally, we might also exploit similar time series (e.g., sales of product $P3$ in region $R2$) and derive the forecasts from a model created over this time series. Each approach leads to a different forecast accuracy and different costs as we have to maintain all models stored in the database.



Query 1 (dashed area)

```
SELECT    time, sales
FROM      facts
WHERE     product = 'P4'
AND       city = 'C4'
AS OF     now() + interval '1 day'
```

Query 2 (dashed + dotted area)

```
SELECT    time, SUM(sales)
FROM      facts
WHERE     product = 'P4'
AND       region = 'R2'
GROUP BY  time
AS OF     now() + interval '1 day'
```

Figure 2: Forecast Queries in Multi-Dimensional Data Sets

Model Configuration Advisor In the style of traditional materialized view and index advisors, we introduce a forecast model advisor that suggests a physical design of forecast models for a given query workload [FSHL13]. The technical challenge of the advisor is the fact that there are no known ways to estimate the accuracy of a physical design of forecast models without actually deploying and querying it. However, deploying a physical design is costly since statistical models can take a long time to be built, especially for a large number of time series.

The model configuration advisor receives a data set and associated query workload as input. It outputs a *model configuration* as well as associated *forecast error* and *model costs*. The advisor 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. 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. To find model candidates, we use *indicators* that heuristically indicate the expected benefit of a model without removing or, more importantly, building it. Then, in the evaluation phase, the benefit of adding or deleting candidate models is empirically evaluated by explicitly creating forecast models and executing the given query workload. Parameters of the advisor like the number of candidate models are automatically tuned in a control phase. Our evaluation shows that our advisor is able to achieve a higher forecast accuracy than alternative approaches and that we are able to calculate a configuration in reasonable time even for larger data sets and complex models.

6 Subscription-Based Forecast Queries

We finally extend forecast queries with continuous aspects, allowing an application (subscriber) to register a query once at F^{2}DB. We call such kinds of queries *subscription-based forecast queries* [FBLP12]. Subscription-based forecast queries arise when an application continuously requires forecast values for further processing. Forecast queries contain the unique characteristic that they can provide an arbitrary number of forecast values, i.e., by adjusting the forecast horizon. 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. The subscriber therefore wishes to be notified only when forecast values have changed relevant to the application.

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, a simple forecast query might request continuous forecasts of energy demand for the next 2 hours with a threshold of 10%. As time proceeds and new time series values arrive, models are updated and optionally maintained by the DBMS. This results in better forecasts leading to notifications sent to the subscriber that contain at least the forecast horizon specified by the subscriber (denoted as *notification length*). In the previous example, a notification needs to be sent if the subscriber holds less than 2 hours of forecast values or 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 depend on the notification length, which further influences the number of notifications. These *subscriber costs* can be communicated to the database system to optimize future notifications.

Our objective is the reduction of the processing costs of the subscriber by trading the number of notifications against the notification length. We approach this challenge by introducing three computational approaches, which are based on the available historical time series data and the cost model of the subscriber. The first one intends to determine a static notification length. The second one determines different notification lengths according to different time slices. Finally, the third one determines the next notification length online. 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 model in real-world situations.

7 Conclusions

We proposed the deep integration of forecasting within a traditional database management system. Declarative forecast queries enable forecasting for any user, especially those not trained in statistics. Arbitrary forecast queries allow the processing of the forecast itself inside the DBMS and the association with other source data (e.g., through joins). Forecast models are handled as first-class citizens and transparently processed inside the DBMS. This allows the materialization and reuse of models by multiple queries as well as transparent query optimization. The introduced novel flash-forward database system opens up many interesting research directions. For example, future work could look at new forecast query optimization techniques, develop parallel forecast operators, or investigate the online maintenance of model configurations.

References

[ACR$^+$11] Mert Akdere, Ugur Cetintemel, Matteo Riondato, Eli Upfal, and Stanley B. Zdonik. The Case for Predictive Database Systems: Opportunities and Challenges. In *CIDR*, pages 167–174, 2011.

[BJR08] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis: Forecasting and Control, 4th ed.* Wiley, 2008.

[CDD$^+$09] Jeffrey Cohen, Brian Dolan, Mark Dunlap, Joseph M. Hellerstein, and Caleb Welton. MAD Skills: New Analysis Practices for Big Data. *PVLDB*, 2:1481–1492, 2009.

[DM06] Amol Deshpande and Samuel Madden. MauveDB: Supporting Model-based User Views in Database Systems. In *SIGMOD*, pages 73–84, 2006.

[FBLP12] Ulrike Fischer, Matthias Boehm, Wolfgang Lehner, and Torben Bach Pedersen. Optimizing Notifications of Subscription-Based Forecast Queries. In *SSDBM*, pages 449–466, 2012.

[Fis14] Ulrike Fischer. *Forecasting in Database Systems*. PhD thesis, Technische Universität Dresden, 2014.

[FRBL10] Ulrike Fischer, Frank Rosenthal, Matthias Böhm, and Wolfgang Lehner. Indexing Forecast Models for Matching and Maintenance. In *IDEAS*, pages 26–31, 2010.

[FRL12a] Ulrike Fischer, Frank Rosenthal, and Wolfgang Lehner. F2DB: The Flash-Forward Database System. In *ICDE*, pages 1245–1248, 2012.

[FRL12b] Ulrike Fischer, Frank Rosenthal, and Wolfgang Lehner. Sample-Based Forecasting Exploiting Hierarchical Time Series. In *IDEAS*, pages 120–129, 2012.

[FSHL13] Ulrike Fischer, Christopher Schildt, Claudio Hartmann, and Wolfgang Lehner. Forecasting the Data Cube: A Model Configuration Advisor for Multi-Dimensional Data Sets. In *ICDE*, 2013.

[GH06] Jan G. De Gooijera and Rob J. Hyndman. 25 Years of Time Series Forecasting. *International Journal of Forecasting*, 22:443–473, 2006.

[GLW$^+$11] Philipp Große, Wolfgang Lehner, Thomas Weichert, Franz Färber, and Wen-Syan Li. Bridging Two Worlds with RICE Integrating R into the SAP In-Memory Computing Engine. *PVLDB*, 4:1307–1317, 2011.

[HKSG02] Rob J. Hyndman, Anne B. Koehler, Ralph D. Snyder, and Simone Grose. A State Space Framework for Automatic Forecasting Using Exponential Smoothing Methods. *International Journal of Forecasting*, 18:439–454, 2002.

[KNR13] Arun Kumar, Feng Niu, and Christopher Ré. Hazy: Making it Easier to Build and Maintain Big-data Analytics. *Queue*, 11(1):30–346, 2013.

[MB98] John T. Mentzer and Carol C. Bienstock. The Seven Principles of Sales-Forecasting Systems. *Supply Chain Management Review*, pages 76–83, 1998.

[MYC05] Boriana L. Milenova, Joseph S. Yarmus, and Marcos M. Campos. SVM in Oracle Database 10g: Removing the Barriers to Widespread Adoption of Support Vector Machines. In *PVLDB*, pages 1152–1163, 2005.

[Sto12] David Stodder. Customer Analytics in the Age of Social Media. The Data Warehousing Institute Research, July 2012.