Malu Castellanos
Umeshwar Dayal
Elke A. Rundensteiner (Eds.)

LNBIP 154

Enabling Real-Time Business Intelligence

6th International Workshop, BIRTE 2012
Held at the 38th International Conference
on Very Large Databases, VLDB 2012
Istanbul, Turkey, August 2012, Revised Selected Papers

Springer

Lecture Notes
in Business Information Processing 154

Series Editors

Malu Castellanos
Umeshwar Dayal
Elke A. Rundensteiner (Eds.)

Enabling Real-Time Business Intelligence

6th International Workshop, BIRTE 2012
Held at the 38th International Conference
on Very Large Databases, VLDB 2012
Istanbul, Turkey, August 27, 2012
Revised Selected Papers

Springer

Volume Editors

Malu Castellanos
Hewlett-Packard
Palo Alto, CA, USA
E-mail: malu.castellanos@hp.com

Umeshwar Dayal
Hewlett-Packard
Palo Alto, CA, USA
E-mail: umeshwar.dayal@hp.com

Elke A. Rundensteiner
Worcester Polytechnic Institute
Worcester, MA, USA
E-mail: rundenst@cs.wpi.edu

ISSN 1865-1348 e-ISSN 1865-1356
ISBN 978-3-642-39871-1 e-ISBN 978-3-642-39872-8
DOI 10.1007/978-3-642-39872-8
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013946091

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

We are delighted to welcome you to the proceedings of the 2013 edition of the BIRTE workshop associated with the Very Large Data Bases Conference (VLDB 2013). The BIRTE workshop series provides a forum in which to discuss and advance the science and engineering enabling real-time business intelligence and the novel applications that build on these foundational techniques. Following the success of our previous workshops co-located with the VLDB conferences in Seoul, Auckland, Lyon, Singapore, and Seattle, our sixth workshop was held in Istanbul, Turkey, in August 2012.

The BIRTE series focuses on issues and challenges related to real-time business intelligence, which has evolved into a multi billion dollar market over the last decade and is continuing to explode with the emergence of big data in all facets of our business and social life. Faced with the emergence of extremely large, diverse, and complex data sets generated at ever-increasing speed, so-called big data, business intelligence systems must effectively analyze these big data to maximally exploit them by drawing deep insights of tremendous business significance. These systems aim to derive the latest understanding of the state of the enterprise as well as of emerging trends – leading to business innovation and advancement in the modern information age. In fact, today's highly dynamic environments require us to provide actionable intelligence practically instantaneously yet over potentially huge data volumes often taking the form of fast data streams.

Moreover, twitter, blogs, and other media means represent a wealth of data on potential customers, their habits and experiences with goods and services in an open framework. Clearly, maximally leveraging this knowledge with minimal latency and utilizing it to drive innovation brings an enormous competitive advantage to a business. Furthermore, businesses are increasingly faced with the need to integrate unstructured information, starting from textual information in corporate intranets and the Web, up to video data streams. The end goal is to support timely decision making, enabled by the availability of instantaneous up-to-date information.

Although there has been progress in this direction, there is still a long way to go. In particular, the whole life cycle of business intelligence requires new techniques and methodologies capable of dealing with the requirements imposed by the new generation of business intelligence (BI) applications. From the capture of real-time business data to the delivery of actionable information, all stages of the BI cycle call for new strategies to tackle these new challenges. These new functionalities may include BI over text data, ensuring information quality, dealing with the uncertainty of prediction models, nested complex events, real-time analytics, and optimizing complex ETL workflows, just to name a few.

BIRTE 2012 featured an exciting technical program, with a highly attended keynote by Donovan Schneider (Salesforce.com) presenting the key concepts of real-time reporting at Salesforce. Donovan explained the issues with running millions of real-time reports every day on the Salesforce multi-tenant cloud platform and presented the solution, including the architecture, query processing, optimization strategies, and other considerations, to getting real-time reporting at such scale. The afternoon session on "ETL and Query Processing in Real-Time BI" started with the invited talk "On-Demand ETL Architecture for Right-Time BI" by Florian Waas from EMC/Greenplum. He outlined some of the key differences practiced in Greenplum for ETL queries in the view of real-time query processing.This session also featured the invited talk by Raymond Ng from the University of British Columbia entitled "Towards Multi-modal Extraction and Summarization of Conversations" that described the challenges in summarizing email and other conversational data and discussed open problems to make it a reality and be conducted in real-time. The final session on "Challenges and Advances in Analytics Platforms" was opened by the invited talk on the topic of the "Live Analytics Service Platform" from HP Labs by Meichun Hsu. In her presentation, Hsu highlighted the challenges, key features, and integration of data management and analytics services in HP Lab's Real-time Analytics Platform.

The program also included a number of excellent peer-reviewed research papers. In the morning session, Ulrike Fischer's talk on "Real-Time Business Intelligence in the MIRABEL Smart Grid System" (TU Dresden and Aalborg University) presented the challenges in real-time BI and the data management solutions for a distributed energy grid system, the MIRABEL project. The second paper on "Data Mining in Life Sciences Using In-Memory DBMSs" by Joos-Hendrik Boese et al. from SAP described how HANA meets some inherent requirements of data mining in life sciences and presented a use case. In the first afternoon session the work by Yagiz Kargin et al. on "Instant-On Scientific Data Warehouses for Data Intensive Research" addressed the inefficiencies in ETL processes commonly used in scientific data analysis by proposing Lazy ETL. In the final session Konstantinos Zoumpatianos (University of Trento) presented the paper "Strategic Management for Real-Time Business Intelligence" that introduced the idea of developing a BI stack on top of data warehouses to enable continuous evaluation of analytical queries on them. In the talk "Pricing Approaches for Data Markets," Alexander Loser (TU Berlin) highlighted the key challenges with regard to pricing strategies for trading raw data, associated analytical services, and analytic results on cloud-based platforms in different market situations. He posed several interesting research problems for the business intelligence community. Overall, there is clearly a wide spectrum of related technologies to be explored and much interest in this area.

We wish to express special thanks to the Program Committee members who helped us prepare an interesting program by producing three reviews per submitted paper. All papers received fair and thoughtful consideration through thorough reviews and discussion. To our keynote and invited speakers, presenters, and attendees, we express our appreciation for sharing their work and the lively

discussions that made this workshop a resounding success. We thank the VLDB 2012 organizers for their continuous support. Finally, we would like to thank Chuan Lei, our Publication Chair, for his efforts in putting these proceedings together and to Medhabi Ray for taking care of the website.

Malu Castellanos
Umeshwar Dayal
Elke A. Rundensteiner

Organization

Organizing Committee

General Chair

Umeshwar Dayal	Hewlett-Packard, USA

Program Committee Chairs

Malu Castellanos	Hewlett-Packard, USA
Elke Rundensteiner	Worcester Polytechnic Institute, USA

Program Committee

Christof Bornhövd	SAP Labs, USA
Ben Chin Ooi	National University of Singapore, Singapore
Howard Ho	IBM, USA
Meichun Hsu	HP Labs, USA
Alfons Kemper	Technical University of Munich, Germany
Christian Konig	Microsoft, USA
Wolfgang Lehner	Dresden University of Technology, Germany
Alexander Loeser	Technical University of Berlin, Germany
Jose Norberto Mazon	University of Alicante, Spain
Renee Miller	University of Toronto, Canada
Torben B. Pedersen	Aalborg University, Denmark
Donovan Schneider	SalesForce, USA
Eric Simon	SAP-BO, France
Nesime Tatbul	ETH Zurich, Switzerland

Proceedings Chair

Chuan Lei	Worcester Polytechnic Institute, USA

Web Chair

Medhabi Ray	Worcester Polytechnic Institute, USA

Table of Contents

Real-Time Business Intelligence in the MIRABEL Smart Grid System

Ulrike Fischer[1], Dalia Kaulakienė[2], Mohamed E. Khalefa[2], Wolfgang Lehner[1], Torben Bach Pedersen[2], Laurynas Šikšnys[2,⋆], and Christian Thomsen[2]

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

Abstract. The so-called smart grid is emerging in the energy domain as a solution to provide a stable, efficient and sustainable energy supply accommodating ever growing amounts of renewable energy like wind and solar in the energy production. Smart grid systems are highly distributed, manage large amounts of energy related data, and must be able to react rapidly (but intelligently) when conditions change, leading to substantial real-time business intelligence challenges. This paper discusses these challenges and presents data management solutions in the European smart grid project *MIRABEL*. These solutions include real-time time series forecasting, real-time aggregation of the flexibilities in energy supply and demand, managing subscriptions for forecasted and flexibility data, efficient storage of time series and flexibilities, and real-time analytical query processing spanning past and future (forecasted) data. Experimental studies show that the proposed solutions support important real-time business intelligence tasks in a smart grid system.

Keywords: BI over streaming data, real-time decision support, tuning and management of the real-time data warehouse, smart grids, renewable energy, flexibility defining data, forecasting.

1 Introduction

Production from renewable energy sources (RES) such as solar panels and wind turbines highly depends on weather conditions, and thus cannot be planned weeks ahead. The EU FP7 project *MIRABEL* (Micro-Request Based Aggregation, Forecasting and Scheduling of Energy Demand, Supply and Distribution) [1] addresses this challenge by proposing a "data-driven" solution for balancing demand and supply utilizing *flexibilities* in time and in energy amount of consumption and production.

Figure 1 illustrates energy loads in the electricity grid before (left side of the figure) and after (right side of the figure) the *MIRABEL* solution balances energy in the grid. The solid gray area depicts non-flexible demand (e.g., usage of

⋆ This work is partially done while visiting Dresden University of Technology.

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 1–22, 2013.

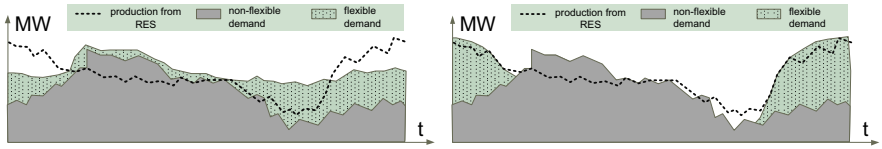
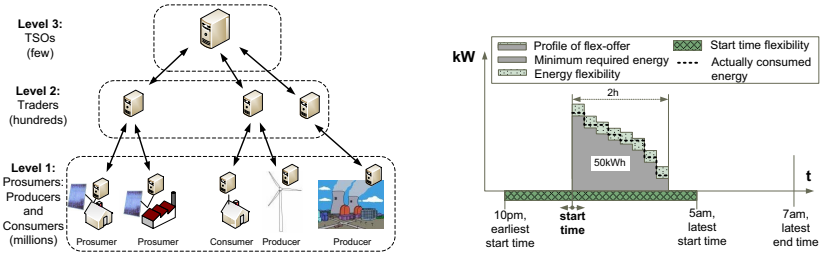


Fig. 1. Balancing consumption and RES production

cooking stoves, lamps, TVs) aggregated from many consumers and varying over some time interval. The dotted area depicts aggregated flexible demand (e.g., usage of washing machines, charging of electric vehicles) that can potentially be shifted to a time when surplus production from RES is available. The dashed line shows available production from RES. The *MIRABEL* project aims to enable the utilization of demand flexibility in time so that higher amounts of the RES production can be consumed (see the right side of Figure 1).

The *MIRABEL* project designs and prototypes an *electricity data management system* (shortly EDMS) that consists of many distributed nodes. These nodes will eventually be deployed at the sites of different actors of the European Electricity Market [2] and their deployment architecture will reflect the hierarchical organization of the European market (Figure 2(a)). Nodes at the lowest level of the system will belong to consumers, producers, and prosumers (entities that both produce and consume electricity), e.g., households, small and large industries. Nodes at the second level will belong to traders, e.g., utility companies or balance responsible parties (BRPs). Finally, the nodes at the highest level will be managed by transmission system operators (TSOs).

The whole EDMS will operate by exchanging and manipulating up-to-date energy related data: (1) time series – measurements of energy consumption and production at each fine-grained time interval, and (2) *flex-offers* – special energy planning objects representing a prosumer's intention to consume (or produce) a certain amount of energy in a specified future flexible time interval. Figure 2(b) shows a flex-offer's energy profile with minimum (solid gray area) and maximum (dotted area) required energy at particular time slots, and the starting



(a) Intended EDMS architecture (b) Flex-offer example

Fig. 2. The intended EDMS architecture and flex-offer for charging a car's battery

time flexibility (shaded area). A flex-offer can also represent demand (or supply) flexibility of a group of prosumers.

In order to meet the project's goals, it is crucial that the EDMS (with all its nodes) is able to efficiently propagate such type of energy data up and down in the hierarchy. Higher level (TSOs, BRPs) nodes must allow storing, querying, forecasting and updating of such data in a timely manner, while lower level (prosumer) nodes might support only the subset of the functionality. In this respect, the *MIRABEL* system is unique due to its real-time requirements, very distributed and hierarchical nature, handling of flex-offers, and the combination of past and future (i.e., predicted) values in querying.

This paper focuses on the data management aspects of a higher level (BRP or TSO) node while leaving those aspect associated with the distribution of nodes for future work. First, it exemplifies the real-time aspects encountered in the EDMS by providing a real-world *MIRABEL* use case. Then, it presents the initial work done designing and implementing a *real-time data warehouse* that operates locally in a node and offers near real-time data management and querying of the two essential entities in *MIRABEL* – time series and flex-offers. The paper thus presents 1) the architecture of the data warehouse, 2) underlying real-time business intelligence challenges, 3) internal warehouse components, and 4) initial results of experiments on individual components.

The rest of the paper is organized as follows. Section 2 presents the use case example of the *MIRABEL* system. Section 3 introduces the data warehouse architecture and the data flow between internal components. The individual components and their underlying real-time challenges are presented in Sections 4–8. Initial experimental results for some components are presented in Section 9, followed by related work in Section 10 and the conclusions and future work in Section 11.

2 MIRABEL Use-Case

In order to demonstrate the typical flow of events in the *MIRABEL* system, we employ the example of charging an electric vehicle utilizing the system.

1. A user of an electric car comes home at 10pm and wants to charge the battery of the car before next morning for the lowest possible price. Once plugged in, the outlet recognizes the car and chooses a default energy consumption profile, according to which the totally required energy is 50kWh and the default charging completion time is 7am.
2. The consumer's node of the EDMS automatically generates a flex-offer (see Figure 2(b)) and sends it to the trader's node of the EDMS where it is aggregated with other similar flex-offers and then scheduled. While taking into account external weather and locally computed energy production and consumption forecasts, the trader's node schedules this particular flex-offer as a part of a larger (aggregated) flex-offer so that charging starts at 1am. This lowers the energy demand peak at 11pm and consumes surplus production of

RES (e.g., wind) at 1am. The trader sends to the consumer node a schedule satisfying the original flex-offer.
3. Simultaneously, the trader's node collects streams of energy measurements to be able to forecast demand and supply in the future. When newly arrived measurements differ substantially from current forecasted values, the respective forecasting model parameters are updated transparently.
4. At 12pm, the TSO notices an upcoming shortfall of supply at 1am–2am and instructs the trader to reduce its demand by a specific amount during this period. The trader, consequently, reschedules the consumer's flex-offer to start energy consumption at 3am and sends an updated schedule back to the consumer's node, also with an updated price of the energy.
5. The consumer;s node starts charging the battery of the electric vehicle at 3am and finishes the charging at 5am.
6. Next month, the trader sends to the consumer an energy bill which reflects the reduced energy costs for user's offered flexibility.

In *MIRABEL*, each flex-offer has two attributes (*assignment-before* and *start-time*) which indicate the latest possible time a flex-offer is allowed to be scheduled and the latest possible time the delivery of the energy has to be started, respectively. These timing attributes of a flex-offer impose that critical real-time constraints must be respected by the EDMS. Consequently, the forecasting, aggregation, and scheduling[1] must start as soon as new data is available and complete before the deadlines imposed by the flex-offer timing attributes values. This is not trivial as a node (at trader or TSO level) must deal with millions of flex-offers (expected number) collected from a few hundred thousands of prosumers. In addition, for a longer term energy planning (day-ahead, week-ahead) or risk analysis, a support for advanced analytics over such volatile data is required. As a consequence, a node of the EDMS must be equipped with an efficient data warehouse to be able to process and analyze queries on energy data in near real-time. These queries can also be on past or future data. To meet these requirements in real-time, our proposed approach stores the underlying time series as models (see Section 4). In the next section, we present the architecture of such an energy data warehouse. Note, we use the term real-time when referring to the "soft" real-time or the near real-time concept.

3 System Architecture

The *MIRABEL* EDMS consists of distributed, non-centralized (although hierarchical) nodes with homogeneous communication interface. In this section, we present the architecture of the single EDMS node and focus on our envisioned energy data warehouse which is based on PostgreSQL.

The *MIRABEL* use-case leads to multiple preliminaries and requirements for the needed warehouse. First, time series and flex-offer data storage and querying need to be supported inherently (among other types of energy data). Large

[1] The reliability of nodes and the latency of communication is not considered in this paper.

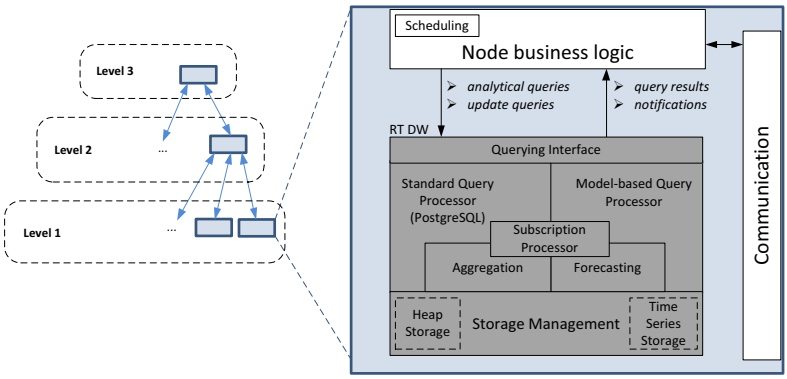


Fig. 3. Internal node architecture

volumes of updates of such volatile data must be efficiently processed. Second, forecasting of time series values must be performed efficiently and the querying of such forecasted values must be supported. Finally, advanced queries offering different processing times and accuracy of results must be enabled. Such queries are needed in *MIRABEL* as various data processing tasks require different precision and different access times of data, e.g., invoicing requires precise and detailed data, but there are no constraints on the processing time (i.e., no near real-time processing is involved). Real-time monitoring, on the other hand, requires fast access time, but tolerate approximate or aggregated results. Thus, queries should provide different results – approximate (trend-analysis, long term energy planning), with some imprecision (grid load balancing), and very precise (billing, risk analysis). In the following paragraphs, we present the *real-time data warehouse* and show how it handles such energy data and queries.

On the right side of the Figure 3, we show the EDMS node architecture. It is composed of a Communication component, the Node Business Logic, and a *real-time data warehouse* (RT DW). In this paper, we discuss issues of the *real-time data warehouse* (depicted in gray). It is a layered system consisting of seven individual components, namely, *Storage Management*, *Aggregation*, *Forecasting*, *Subscription Processor*, *Standard Query Processor*, *Model-based Query Processor*, and *Querying Interface*. The *real-time data warehouse* receives analytical and update queries and provides query results as well as notifications to the node business logic. Part of this business logic is the *Scheduling* component (see more details in Tušar et. al. [3]), which implements a particular flex-offer scheduling strategy depending on the business goals of the market player (owner of the node). The node business logic communicates with the other nodes to receive and propagate data such as measurements, flex-offers, or schedules.

Storage Management provides efficient storage for energy data including flex-offers and time series, such as weather data as well as consumption and production measurements. This component is equipped with an intermediate main-memory data store, which accumulates arriving data, makes it available for queries, and materializes it later on (in standard PostgreSQL *Heap Storage*).

Additionally, *Storage Management* stores time series models and their parameters (in *Time Series Storage*). *Aggregation* is responsible for incrementally and efficiently aggregating multiple flex-offers into fewer ones with larger energy amount values. Based on time series values, *Forecasting* builds and maintains forecast models and makes future values of time series available for queries. *Subscription Processor* monitors *Aggregation* and *Forecasting* and notifies subscribers when aggregated flex-offers or forecast values change substantially (compared to previous notifications). There are two query processors employed by our data warehouse: *Standard Query Processor* and *Model-based Query Processor*. *Standard Query Processor* (PostgreSQL) accesses *Heap Storage* and provides exact results for user-issued queries. Such queries might be complex and long-running and might involve aggregated and non-aggregated data such as flex-offers and time series. *Model-based Query Processor* relies on historical and future time series models (stored in *Time Series Storage*) and allows approximate but efficient answering of historical, future, and combined queries. Finally, *Querying Interface* offers a unified interface for queries and query results.

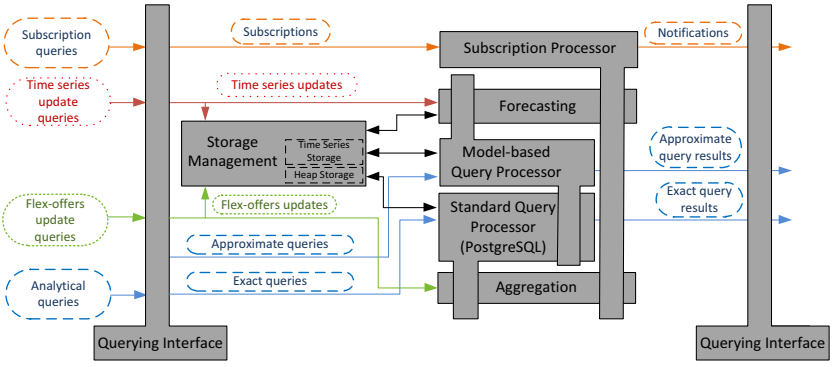


Fig. 4. Data flow during query processing in the real-time data warehouse

Figure 4 visualizes the flow of the queries and the data during query processing in the *real-time data warehouse*. Each query type is first processed by *Querying Interface* and then handed to the appropriate components. The flow path of queries and data differs depending on the type of queries being executed in the data warehouse:

Subscription queries are handled by *Subscription Processor*. Such queries allow users to (un-)subscribe (from) to notifications that are generated every time results of a user-specified continuous query changes more than a specified threshold. Two important types of continuous queries (among others) are supported by our data warehouse: flex-offer aggregation and time series forecasting queries. One example of a subscriber is the *Scheduling* component which is only informed when forecasts or aggregated flex-offers change substantially.

Time series update queries (value inserts and deletes) are passed to *Storage Management* and *Forecasting* simultaneously. The *Storage Management* component handles the materialization of these updates, while the *Forecasting* component uses these updates to incrementally maintain existing forecasting models. Those models are monitored by *Subscription Processor*.

Flex-offer update queries are processed similarly to the time series update queries and passed to *Storage Management* and *Aggregation* simultaneously. If aggregated flex-offers change substantially, a notification is generated by *Subscription Processor*.

Analytical queries can be either exact or approximate queries. Exact queries are routed to *Standard Query Processor*, while approximate queries are passed to *Model-based Query Processor*. While processing exact queries, the *Standard Query Processor* component accesses *Heap Storage* and can take advantage of *Aggregation* to answer analytical queries over flex-offers. Similarly, *Model-based Query Processor* can utilize *Forecasting* to answer queries on future (or both historical and future) data from *Time Series Storage* and utilize *Standard Query Processor* to answer an exact part of the queries.

In the following sections, we discuss individual components of *real-time data warehouse* in more detail.

4 Storage of Energy Related Data

In this section, we focus on the *Storage Management* component and discuss flex-offer and time series storage as well as data loading.

4.1 Flex-Offer Storage

To store flex-offer data, we employ a multi-dimensional schema that includes several *fact* and *dimension* tables (more details about the schema can be found in Šikšnys et. al. [4]). The dimension tables represent time intervals, metering points, possible states of flex-offers as well as legal entities. The fact tables represent flex-offers themselves. The schema is designed to enable convenient storage of energy demand and supply flexibilities and efficient processing of the most common analytical queries in the *MIRABEL* system.

Essentially, there are two fact tables called *F_flexOffer* and *F_enProfileInterval*. They hold flex-offer facts and information about energy profiles (composed of multiple intervals) of a flex-offer, respectively. Each stored flex-offer is given a unique identifier. For each stored profile interval, there are measures to specify the duration of the profile interval as well as the lowest and highest amount of energy needed. The *F_flexOffer* table holds information about time deadlines associated with flex-offer, i.e., *enProfile_startAfter* time for initial or *enProfile_startFix* time for scheduled flex-offer. As flex-offers can be aggregated into larger flex-offers, we also have the table *F_aggregationMeta* which references *F_flexOffer* twice to point to the aggregated "parent flex-offer" and the smaller "child flex-offer" which has been aggregated, respectively.

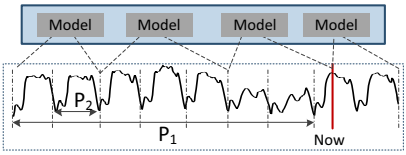


Fig. 5. TimeSeries with associated models

Our proposed schema allows to read and write flex-offers efficiently, which is important for flex-offer aggregation as well as other analytical queries, e.g., those used in the energy balancing.

4.2 Time Series Storage

We employ different time series storage schemes which are utilized in the processing of approximate queries with allowed impressions and exact queries with the precise result.

For exact queries, we utilize standard PostgreSQL *Heap Storage* to store the original time series as a block-based array. The benefits of storing the time series as a block-based array are as follows: (1) it alleviates the need for sorting the time series, e.g., when optimizing forecast model parameters, (2) it eases the access to values as only relevant pages (blocks) with requested values have be accessed when answering a query. To determine the range of needed pages, the WHERE clause of a query can be utilized.

For approximate queries, we compactly represent time series over past and future using models and put them in the *Time Series Storage*. To achieve this, we build a set of models that incorporate seasonal and non-seasonal behavior. As time proceeds, the system incrementally updates the models and utilizes them to answer approximate queries. To find the suitable set of models, we divide the time series into non-overlapping sub-intervals (e.g., using approaches in [5,6]). To build a model over the interval, we decompose the portion of time series over this interval into *trend*, *seasonal*, and *remainder* components utilizing the information about the seasonal periods. A time series can have an arbitrary number of seasonality periods (e.g., the time series shown in Figure 5 exhibits the two seasonality periods P_1 and P_2). Each model represents a segment of the underlying time series with the predefined error guarantee (top of Figure 5). For query optimization, we store statistics about the index in the system catalog. Specifically, we store the number of models segments and their average size. These statistics are needed to estimate the expected cost for the approximate query.

As explained in our previous work [7], models built for each portion of the time series can also be composed into a hierarchical structure, where models at different levels represent a time series with a certain (bounded) precision. To summarize, time series are stored in the data warehouse as block-based arrays and also as models that represent parts of time series at various precisions.

4.3 Data Bulk Loading

Flex-offers as well as values of time series are initially cached before they are physically stored on disk. First, arriving data is recorded in an intermediate data store in main memory similar to RiTE [8] such that the data is made available for querying before it gets materialized on the disk. Finally, the data is physically inserted into the underlying data warehouse in bulks.

The storage of the data, might, however, still be slow due to high update rate (that might occur sporadically). Instead, for time series, measured values can be ignored if forecast models predict the value within the bounded imprecision. If not, the value can be used to update the forecast model while storing only new model parameters in the data warehouse. Since only models are stored, there are less data values to store and the processing is much faster.

The main advantage of the latter approach is that it supports fast (although imprecise) data loading that enables approximate answering of analytical queries. It also allows transparent processing both of queries that need only fast and approximate answers and of queries that need more accurate answers since all the data can be made available in forecast models. Further, the model-based storage can lead to compression of the data which in turn means that the data may fit in main memory.

5 Real-Time Flex-Offer Aggregation

Flex-offer aggregation is triggered when a consumer sends a new flex-offer or trader wants to reschedule older flex-offers (see Section 2). The *Aggregation* component combines a set of many *micro* flex-offers into a set of fewer *macro* flex-offers. It is a complex operation as, even for two flex-offers, there are many different ways to combine them into one flex-offer. Each of these combinations results in different flexibility loss, which often needs to be minimized. Fortunately, by grouping flex-offers carefully, it is possible to aggregate many flex-offers efficiently (in linear time) while still preserving lots of their flexibilities [9]. As presented in Section 2, a very large number of flex-offers need be scheduled in *MIRABEL*. Since scheduling is an NP-complete problem [10], it is infeasible to schedule all these flex-offers individually within the (short) available time. Instead, flex-offers should be aggregated first, then scheduled, and finally *disaggregated*. In this process, the additional disaggregation operation transforms *macro* scheduled flex-offers into *micro* scheduled flex-offers. As the aggregation plays a major role during the scheduling, we focus on real-time aspects in this particular use-case of the aggregation in the following sub-sections.

5.1 Real-Time Aggregation Challenge

In *MIRABEL*, a higher level (e.g., BRP) node receives a continuous stream of flex-offers from many lower level (e.g., prosumer) nodes. Simultaneously, execution deadlines of the previously received flex-offers are approaching and thus

their respective individual schedules (assignments) have to be prepared and distributed back to the original issuers of the flex-offers. When aggregating those flex-offers, a number of requirements have to be satisfied [9, 11]:

Disaggregation requirement. Aggregation must ensure that it should be always possible to correctly disaggregate scheduled macro (aggregated) flex-offers into scheduled micro (non-aggregated) flex-offers while matching two schedules produced before and after disaggregation. Here, two schedules match when total energy amounts at every time interval are equal.

Compression and flexibility trade-off requirement. Aggregation should allow controlling a trade-off between the number of aggregated flex-offers and the loss of flexibility, i.e., the difference between the total time/amount flexibility before and after the aggregation.

Aggregate constraint requirement. It must be possible to bound the total amount defined by every aggregated flex-offer, e.g., in order to meet power grid constraints.

After flex-offers have been aggregated, they have to be scheduled, and disaggregated thus producing individual schedules that can be distributed back to the lower level nodes, i.e., prosumers. In *MIRABEL*, the total time available for the complete process of flex-offer aggregation, scheduling, and disaggregation is 15 minutes. This deadline should not be missed as a new set of flex-offers and new forecasts are expected to be available after this period of time. There is a challenge to build such a flex-offer aggregation solution which would satisfy all above-mentioned requirements as well as be scalable enough to deal with millions of flex-offers while still providing a sufficient amount of time for scheduling and disaggregation (from the totally available 15 minutes interval).

5.2 Real-Time Aggregation Solution

Considering frequently changing flex-offers in the *MIRABEL* system, we build an efficient incremental flex-offer aggregation solution [9], which satisfies all the above mentioned requirements and aims to address the real-time challenge.

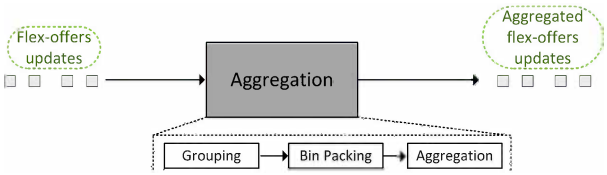


Fig. 6. Input and output of the incremental aggregation solution

First, the disaggregation requirement is inherently satisfied by the solution, because the aggregated flex-offer always defines less (or equal) flexibilities compared to the original flex-offer and thus it is always possible to disaggregate scheduled flex-offers. Second, we introduce the set of aggregation parameters,

and different values of these parameters allow controlling the trade-off between compression and the flexibility loss. Third, we introduce an intermediate *bin-packing* step that ensures that a user-selected constraint, e.g., total maximum amount is below 1000 kWh, is satisfied for every aggregated flex-offer. Finally, our aggregation solution inherently supports incremental aggregation, which provides a huge saving in time when aggregating a flex-offer set that gradually changes in time (as described above). When aggregating flex-offers incrementally, micro and macro flex-offer sets are maintained. As visualized in Figure 6, our incremental solution takes the sequence of micro flex-offer updates as input and produces the sequence of macro (aggregated) flex-offer updates as output. An update in the input contains a flex-offer and indicates whether the flex-offer is *added* or *removed* to/from the micro flex-offer set. Our solution can process such updates one-by-one at a time or in bulks. When the bulk of updates (or a single update) is processed in the *Grouping*, *Bin-packing*, and *Aggregation* steps, the solution updates the set of macro flex-offers and outputs those aggregated flex-offers that either were *added*, *removed*, *modified* during this process. Such incremental behavior is the key addressing the real-time challenge as it allows efficiently processing flex-offers received from lower level nodes as well as those with elapsed execution deadlines. Later in the experimental section, we evaluate the throughput of updates which can be handled by our incremental aggregation solution when one or more updates are processed at a time.

As future work, the support for other types of flexibilities will be provided, e.g., a *duration flexibility*, which allows a BRP to supply a certain amount of energy to prosumers in the preferred duration of time.

6 Real-Time Time Series Forecasting

The *Forecasting* component enables predicting the future development of time series data. In *MIRABEL*, forecasts of energy production and consumption time series are crucial in order to enable the scheduling of aggregated flex-offers (see step 3 of the *MIRABEL* use-case in Section 2). Additionally, the *Forecasting* component needs to efficiently process new energy measurements to detect changes in the upcoming energy production or consumption and to enable the rescheduling of flex-offers if necessary.

6.1 Overview Time Series Forecasting

Sophisticated forecasts require the specification of a stochastic model, a so-called *forecast model*, that captures the dependency of future values on past values. Specific characteristics of energy time series like multi-seasonality (daily, weekly, annual) or dependency on external information like weather or calender events require the employment of forecast models tailor-made for the energy domain [12], [13]. Such models involve a large number of parameters leading to high model creation times due to expensive parameter estimation. Thus, models are pre-computed and stored in the database to support real-time answers.

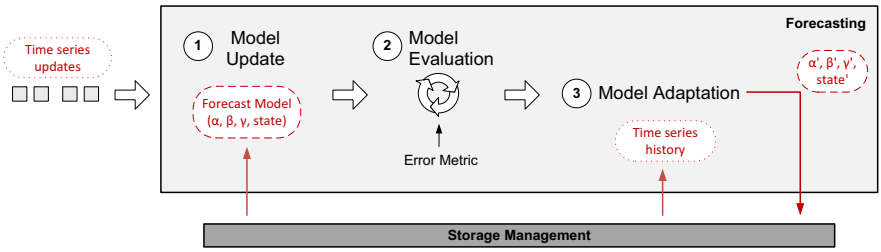


Fig. 7. Overview of forecast model maintenance

However, a continuous stream of new measurements of energy production and consumption leads to evolving time series and thus requires continuous maintenance of the associated forecast models (Figure 7) [14]. For each new measure value, we need to initiate (1) model update, which consists of an incremental state adaption of the forecast model. Then, (2) model evaluation checks the forecast model error with regard to the new measurements. Finally, (3) model adaption might be triggered in order to re-estimate the model parameters and, therefore, adapt the forecast model to the changed time series characteristic. Model adaption is as expensive as the initial model creation since most parameters cannot be maintained incrementally.

As the calculation of forecast values from existing models can be done very efficiently, model maintenance exhibits the major challenge in terms of real-time processing. Here, we face two main challenges: (1) when to trigger forecast model maintenance and (2) how to efficiently adapt the forecast model parameters. Both aspects strongly influence the amount of time spent on maintenance as well as the forecast accuracy. In the following, we further detail these main challenges of model maintenance and sketch different solutions that utilize standard as well as energy-domain-specific techniques.

6.2 Real-Time Model Adaption

Model adaption might be triggered periodically by a fixed interval, either after a fixed number of new measurements or after a fixed time interval. However, these strategies exhibit the problem of determining the model adaptation interval. Too short intervals lead to unnecessary adaptations of well-performing forecast models, while too long intervals lead to the problem of unbounded forecast errors in the sense that arbitrarily large errors are possible between the defined points of model adaptation. The other strategy is to utilize the continuous stream of consumption and production measurements to continuously evaluate the forecast model accuracy and trigger model adaptation when the forecast error violates a previously defined threshold. While this strategy guarantees that a certain forecast error is not exceeded, it exhibits a similar drawback as the fixed interval model adaptation since it also depends on the definition of suitable thresholds.

To weaken the disadvantages of a single technique, we can combine both evaluation approaches and trigger model adaption time- and error-based.

Model adaption re-estimates the model parameters and therefore, adapts the forecast model to the changed time series characteristics. There, the d parameters of a forecast model span a logical d-dimensional search space, in which we want to find the global optimal solution with regard to the forecast error. In this context, two challenges arise, first, where to start the parameter re-estimation process and, second, how to efficiently re-estimate the parameters itself.

Our core assumption of start point determination is that forecast model parameters will not change abruptly but will be in close neighborhood of previous model parameters. Thus, we should exploit context knowledge from previous model adaption by either using the parameters of the outdated model or a combination of a set of older models. For the last approach, we store previous models of a time series in conjunction with their context information (e.g., weather information) in a repository. If a similar context occurs and maintenance is triggered, we reuse these models to calculate suitable start points [15].

To actually determine the new model parameters, arbitrary global and local optimization methods, each with advantages and disadvantages, can be utilized. Global optimization methods might find a global optimum but need more time to terminate, while local optimization methods follow a directed approach and therefore converge faster to a solution but exhibit the risk of starvation in local sub optima. Again, we can combine both approaches to weaken their disadvantages by starting with local optimization method and using a global optimization method afterwards if there is still time available.

Additionally, we have developed several approaches that utilize characteristics of energy-specific forecast models to speed up the parameter estimation process itself. First of all, energy demand models often involve multi-equation models [13] that create a different forecast models for different time intervals (e.g., for each hour of the day). As multi-equation models consists of several independent individual models, we reduce the parameter estimation time by partitioning the time series and parallelizing the estimation process [16]. Second, energy demand and supply models often require the inclusion of many external sources (e.g., weather information) to increase the accuracy. The naive approach of adding each external time series directly to the forecast model highly increases the model maintenance time due to the large parameter space. Therefore, our approach separates the modeling of external sources from the actual forecasting model and combines both models after parameter estimation.

Up to now, we have only discussed how to efficiently maintain a forecast model for a single time series. However, we might also exploit the hierarchical organization of the data within one node or over several nodes to reduce the overall number of forecast models and thus the maintenance overhead [17].

7 Subscription Processor for Forecasts and Flex-Offers

Within *MIRABEL*, forecasts and aggregated flex-offers are continuously required by scheduling and, thus, might be continuously polled from the data warehouse

in each interval. This is very inefficient if flex-offers and/or forecasts changed only marginally leading to high application costs. To prevent the polling, continuous queries can be registered in the data warehouse leading to notifications to subscribers every time when results of these queries change substantially (due to data changes in the warehouse). Such subscriptions and notifications are handled by the *Subscription Processor* component. In the following sections, we provide examples of continuous flex-offer aggregation and forecasting queries and explain how the respective notifications are generated.

7.1 Flex-Offer Notifications

Continuous flex-offer aggregation queries must define an error threshold, which define a condition for a notification. For example, the following continuous aggregation query requests aggregated flex-offers for the upcoming day specifying the error threshold of at most $10kWh$:

```
SELECT *
FROM AggregateFlexoffers (
     SELECT flexOfferId FROM F_flexOffer
     WHERE enProfile_startAfter = now() + INTERVAL '24 hours')
NOTIFY ON energy_error > 10000
```

A notification is sent to the subscriber when inserts and deletes of flex-offers cause that the result of aggregated flex-offers changes higher than $10kWh$ in the terms of energy amount. A subscriber can specify whether the notification should include the new set of aggregated flex-offers or only the difference compared to the previously reported aggregation result. The proposed flex-offer notification schema implements data pushing rather than pulling approach, thus it prevents the repeated aggregation of the same set of flex-offers in cases when up-to-date aggregates are required (which is common in the *MIRABEL*). Furthermore, they can lower node's communication costs associated with the propagation of flex-offers throughout the hierarchy of the EDMS.

7.2 Forecast Notifications

Similarly to continuous aggregation queries, subscription-based forecast queries can be utilized. The following continuous forecast query requests forecasts for at least the next 12 hours providing an error threshold of 10%.

```
SELECT hour(time), SUM(energy) energy_per_hour
FROM TS_PowerConsumption GROUP BY hour(time)
FORECAST energy_per_hour ON hour(time)
NOTIFY ON MIN HORIZON '12 hours' AND THRESHOLD 0.1
```

For this query, a notification is sent to the subscriber if either (1) time has exceeded and the subscriber holds forecast values for less than 12 hours (time-based) or (2) if new forecasts are available that deviate by more than 10% from old forecasts sent before (threshold-based). As the subscriber only specifies a

minimum number of forecast values, the system can internally decide how many values to send and thus regulate costs. If only a small number of forecast values are propagated, many time-based notifications occur. In contrast, propagating a larger number of forecast values leads to a higher forecast error and thus more threshold-based notifications. Thus, the forecast horizon can either be chosen manually by the subscriber or automatically [18] to ensure accurate forecasts and smaller number of notifications.

8 Query Processing

The real-time data warehouse has two types of query processors called *Standard Query Processor* and *Model-based Query Processor*, respectively. In the following sections, we elaborate the types of queries these two processors support and provide examples of the most important query types in our system.

8.1 Standard Query Processing

Standard Query Processor supports both simple (point, range) and analytical (aggregate, join) queries over historical time series and flex-offer data. All tuples (from *Heap Storage*) relevant for a query have to be retrieved from the main-memory or disk storage in order to process the query, hence processing complex analytical queries over large datasets (of time series or flex-offers) might take a long time. To improve performance, a user can explicitly require the flex-offer aggregation in a query. In this case, *Standard Query Processor* might take advantage of materialized aggregated data (e.g., from *Aggregation*), thus lowering not only data access costs, but also computational costs (e.g., when exploring the flexibility space of flex-offers). In all these cases, the result of a query is always exact. To implement *Standard Query Processor*, a traditional query processor (PostgreSQL) can be utilized and extended with additional user defined functions specific for handling energy data such as flex-offers.

We now give some examples of queries handled by *Standard Query Processor*. The query below shows how the balance between produced and consumed energy at the granularity of 15 minutes can be computed for January of 2012:

```
SELECT hour_quarter(time), SUM(p.energy) AS Prod,
        SUM(c.energy) AS Cons, SUM(p.energy - c.energy) as Balance
FROM TS_PowerProduction p, TS_PowerConsumption c
WHERE year(time) = 2012 AND month(time) = 1
GROUP BY hour_quarter(time)
```

The next query finds maximum and minimum energy amounts that potentially can be scheduled at every time interval (of 15 minutes) utilizing all unscheduled flex-offers:

```
SELECT MAX(en_high) as MaxEnergy, MIN(en_low) as MinEnergy
FROM forAllFlexofferTimeShifts(
       aggregateFlexoffers(
```

```
          SELECT flexOfferId FROM F_flexOffer
          WHERE enProfile_startFix is not NULL))
GROUP BY timeIntervalId
```

Here, the function *aggregateFlexoffers* aggregates flex-offers with ids specified by a sub-query; for every flex-offer, the function *forAllFlexofferTimeShifts* enumerates its possible alternatives to fix (to schedule) flex-offers in time. The latter function outputs several alternatives containing the minimum (`en_low`) and maximum (`en_high`) energy amounts for each time interval (`timeIntervalId`). Enumeration of such alternatives might be very expensive if aggregation is not performed. Thus, the applied aggregation operation substantially improved the performance of such query. Materialized results of flex-offer aggregation might further improve the performance of such queries.

8.2 Model-Based Query Processing

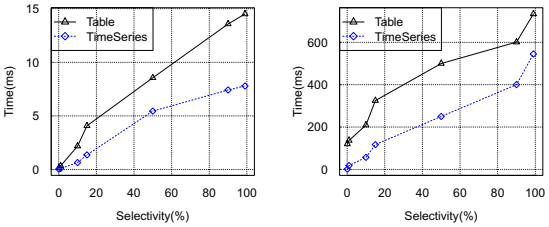
Model-based Query Processor is designed for time series, primarily. It operates on time series models (*Time Series Storage*) rather than tuples. It is usually faster than *Standard Query Processor* as less data needs to be accessed on disk. Furthermore, queries supported by the processor can be executed on both historical and predicted data thus allowing to look into the history and future. To query historical data, a user may explicitly specify error thresholds (absolute error and confidence) that describe how large impressions are allowed to be. Smaller values of the threshold leads to longer query execution times as a larger number of models have to be retrieved from storage (see Section 4.2). If historical models cannot guarantee a required precision, the *Heap Storage* is accessed through *Standard Query Processor*. To query forecasted data, the functionality of *Forecasting* is utilized.

For example, the following query finds the energy consumption of households for the previous 24 hours and upcoming 2 hours with max 5% error:

```
SELECT hour(time), SUM(energy) AS energy_per_hour
FROM TS_PowerConsumption
WHERE category='household' AND
      time BETWEEN (now() - INTERVAL '24 hours') AND
                   (now() + INTERVAL '2 hours')
GROUP BY hour(time) ORDER BY hour(time)
RELATIVE_ERROR 0.05;
```

Such type of queries focusing on recent history and near future is important in real-time monitoring and intra-day planning of energy. In this process, the timely delivery of the query result is vital.

To summarize, two query processors are supported by the real-time data warehouse. A user might chose to execute his query on either of them, depending on the available time for query execution and the required precision of the result.



(a) Accessing Timeseries (b) Overall Performance

Fig. 8. Time Series Storage

9 Experimental Results

We now present some initial experimental results on the storage, aggregation and forecasting components. No other smart grid system does what the *MIRABEL* system does, e.g., with respect to flex-offers, and thus we do not compare with other systems.

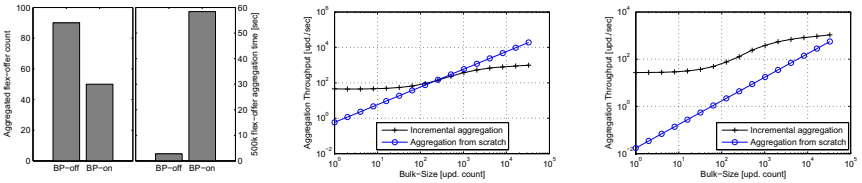
9.1 Storage Experiments

We evaluate query performance for the storage component presented in Section 4.2 using real-world time series dataset of 110,000 tuples. These experiments were run on a computer with Intel CoreI-7 with 8 GB of RAM, and Ubuntu 2.6 (x86 64) OS. In this set of experiments (Figure 8), we compare the performance of representing the time series as a database table (solid line) and natively as a disk-block array (dotted line) by measuring the query time for: (a) a simple query which only accesses the materialized time series and (b) the overall time for accessing the time series and building the forecast model. We increase the selectively of the query from 0.1% to 100% of the dataset.

In Figure 8(a), we achieve 4.5 times speed up for small range queries (i.e., selectively of 0.1%) and up to 1.8 times speed up for almost the entire dataset by using time series represented as arrays. It should be noted that small range queries are typical in the *MIRABEL* scenario of a real-time smart grid application. This is achieved by keeping the heap organized, only accessing the needed pages and minimizing the disk bandwidth. Figure 8(b) shows a higher speed up by eliminating sorting the time series data. More precisely, we achieve 100 times speed up for small range queries (i.e., selectively of 0.1%) to 1.3 times speed up for almost the entire dataset.

9.2 Aggregation Experiments

We evaluate the throughput of flex-offer updates in the incremental aggregation solution. We use the same experimental setup as in our previous two works [3,9]. Here, the dataset [3] represents synthetically generated energy consumption requests issued for a single day from multiple consumers, and the aggregation



(a) Aggregated flex-offer counts and aggregation times (b) Throughput of updates, BP-off (c) Throughput of updates, BP-on

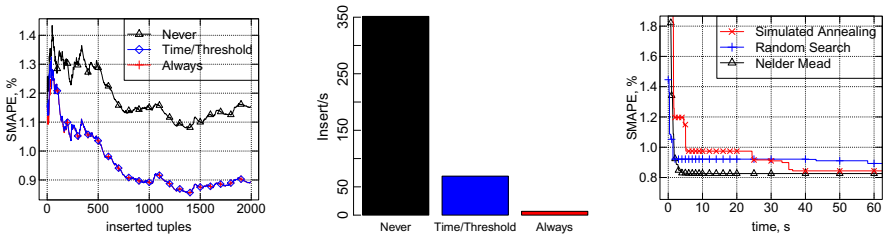
Fig. 9. Flex-offer Aggregation

parameters are set so that the best scheduling result is obtained. In the experiment we initially aggregate 500000 flex-offers. Then, we continuously feed new flex-offers (for the same 24 hours interval) and remove existing flex-offers while keeping the total number of micro flex-offers equal to 500000. We process such inserts and deletes in bulks. When the bulk of a certain size is formed, we trigger the aggregation. Here, the size of the bulk determines the level of aggregation result up-to-dateness, i.e., small bulks yield up-to-date results while larger bulks yield slightly outdated results. Furthermore, we evaluate our approach with bin-packing (BP) feature on and off. The BP-on means that macro flex-offers are guaranteed to have the time flexibility of no less than 2 hours.

Figure 9(a) visualizes the 500000 flex-offer aggregation performance. When the BP is off, 500000 micro flex-offers are aggregated into 90 macro flex-offers. When the BP is on, the number of macro flex-offer is only 50 as some of the original flex-offers are excluded since they result in macro flex-offers with time flexibility lower than 2 hours (which is not allowed by the aggregate constraint). Times spent aggregating flex-offers with BP-off and BP-on are 3 and 58 seconds, respectively. Figure 9(b) shows the throughput of updates our aggregation solution (with BP-off) can handle when 1) incrementally maintaining macro flex-offers with the bulks of flex-offer updates (line with crosses); 2) aggregating 500000 flex-offers from scratch when a new bulk arrives (line with circles). Similarly, Figure 9(c) shows the throughput when the BP is on. As seen in the figures, the throughput of updates increases when we process updates in larger bulks (applies for both incremental and the from-scratch aggregation, BP-on and -off). For small bulk sizes, the incremental aggregation offers better throughput (even of two orders of magnitude). However, when the bulks are very large (few hundreds of updates in the BP-off case) the better throughput is achieved when aggregating flex-offers from scratch on each bulk of updates. In general, to handle the workload of a particular update rate and to provide as fresh aggregation results as possible, an appropriate bulk size has to be chosen and either incremental or the from-scratch aggregation has to be applied.

9.3 Forecasting Experiments

We evaluate the influence of different model evaluation and adaption strategies (discussed in Section 6) using a publicly available energy production data set [19].



(a) Model evaluation - forecast accuracy (b) Model evaluation - maintenance time (c) Model adaption

Fig. 10. Influence of Maintenance Strategies

It consists of a single aggregated wind energy time series from 2004 to 2006 in 10 min resolution. The experiment is conducted using Holt-Winters Triple Seasonal Exponential Smoothing, a forecast model tailor-made for the energy domain [12]. To measure the forecast accuracy, we use the symmetric mean absolute percentage error (SMAPE), which is a scale-independent accuracy measure and takes values between 0% and 100%. The following experiments were executed on an IBM Blade (Suse Linux, 64 bit) with two processors and 4 GB RAM.

In a first experiment, we explore the influence of different model evaluation strategies. A first model evaluation strategy *never* adapts the model parameters, while a second strategy *always* adapts the forecast model after each new measurement. Finally, our third strategy triggers model adaption based on a combined time- and threshold-based approach. For this last approach, we set the time limit to 24 hours and the threshold to 2.5%. Figure 10(a) shows the development of the model accuracy for a series of inserts on the time series for each of the three strategies, while Figure 10(b) shows the estimated number of inserts per second we can achieve with the corresponding strategy. First of all, we can observe that model adaption always leads to a lower forecast error compared to no adaption at all and thus is required to achieve the best possible accuracy. However, if we adapt the model after each new measurement we achieve a much lower throughput than a strategy that never adapts the model parameter. In comparison, our time- and threshold-based approach still leads to a low forecast error but increases the throughput.

In a second experiment, we examine a single model adaption step more closely by comparing the error development of three important parameter estimation strategies (Figure 10(c)). In detail, we measure the accuracy over time of a single adaption step using Simulated Annealing, Random Search and Random Restart Nelder Mead. As it can be seen, all algorithms converge to a result having similar accuracy, where Random Restart Nelder Mead requires much less time than, for example, Simulated Annealing. Thus, the model adaption strategy has a high impact on the required time and needs to be set accordingly.

10 Related Work

In this section, we list other similar smart grid projects, and present work related to the data management components of the *MIRABEL* EDMS.

There are hundreds of ongoing and finished smart grid projects only in Europe [20] with budgets over 3.9 billion euro in total. The most notable are: ADDRESS (25 partners), BeyWatch (8 partners), E-Energy Project "MeRegio" (6 partners), Energy@home (4 partners), INTEGRAL (9 partners), Internet of Energy (40 partners), SmartHouse-SmartGrid (6 partners). The listed ones aim to specify and prototype data exchange infrastructures to demonstrate active customer involvement in the electricity grid operation. The *MIRABEL* project is unique for the "data driven" approach of customer involvement, since the other projects either require end-consumers to negotiate the price and thus actively participate in the electricity market, or are oriented to the distributed energy producers and grid operators. Further, the *MIRABEL* project introduced the flex-offer concept and its precise specification [21] and is now seeking to standardize it [22]. See Section 1 for a discussion of *MIRABEL*'s unique characteristics.

The *MIRABEL* approach exhibits multiple real-time business intelligence requirements and no single system category can cover all of them. Event engines [23], for example, do not support complex analytical queries and advanced analytics (e.g., time series forecasting). Pure data stream management systems (e.g., [24]) can cope with high-rate input streams and rather complex queries; however, ad-hoc data analysis and advanced analytics on historical data are impossible. From the more traditional DBMS side, read-optimized data organization such as column-oriented data management or hybrid OLTP/OLAP solutions [25] provide efficient ad-hoc data analysis but exhibit drawbacks with regard to write-intensive applications. In the data warehouse area, real-time data warehouse approaches try to cope with a continuous stream of write-only updates and read-only queries at the same time [26]. However, the unique concept of flex-offers as well as the requirement of continuous time series forecasting demand an architecture tailor-made for the *MIRABEL* system.

11 Conclusions and Future Work

We have described the real-time business intelligence aspects of the *MIRABEL* system that facilitates a more efficient utilization of renewable energy sources by taking benefit of energy flexibilities. Within *MIRABEL*, real-time advanced analytical queries (including forecasting) have to be supported on a continuous stream of fine-granular time series data and a large number of flex-offers. To cope with these requirements, we introduced a real-time data warehouse solution consisting of the following components. First, the storage management component stores flex-offer and time series data as well as models for historical and future data. With these models, we can provide (1) accurate forecasts of future data, (2) fast approximate results of historical data and (3) increased performance of data

loading. Second, the aggregation component efficiently handles a large number of flex-offers by incrementally grouping similar ones with minor flexibility loss. Third, the forecasting component provides accurate forecasts in real-time due to efficient maintenance of energy-domain specific forecast models. Furthermore, the subscription service for flex-offers and forecasts is provided and the standard query processor is extended to support not only exact queries but also approximate queries on past as well as future data. Preliminary experiments on storage management, aggregation, and forecasting components show the applicability and efficiency of the corresponding approaches.

The remaining challenges for the future work are the following: (1) a distribution of the EDMS nodes, e.g., how to ensure consistency and efficiency and lower the amount of communication when propagating the data, (2) a support more types of flexibilities, e.g., price flexibility, and extend data storage management and aggregation components accordingly, (3) a tight integration of forecasting and data loading techniques to enable real-time data warehousing on massive amounts of data in distributed nodes.

With our proposed solution, we take an important step towards the integration of larger amounts of distributed generated RES into the electricity grid.

Acknowledgment. The work presented in this paper has been carried out in the MIRABEL project funded by the EU under the grant agreement number 248195.

References

1. The MIRABEL Project (2012), `www.mirabel-project.eu/`
2. ETSO, The harmonized electricity market role model (2009), `https://www.entsoe.eu/fileadmin/user_upload/edi/library/role/role-model-v2009-01.pdf`
3. Tušar, T., Šikšnys, L., Pedersen, T.B., Dovgan, E., Filipič, B.: Using aggregation to improve the scheduling of flexible energy offers. In: Proc. of BIOMA (2012)
4. Šikšnys, L., Thomsen, C., Pedersen, T.B.: MIRABEL DW: Managing Complex Energy Data in a Smart Grid. In: Cuzzocrea, A., Dayal, U. (eds.) DaWaK 2012. LNCS, vol. 7448, pp. 443–457. Springer, Heidelberg (2012)
5. Thiagarajan, A., Madden, S.: Querying continuous functions in a database system. In: Proc. of SIGMOD, pp. 791–804 (2008)
6. Shatkay, H., Zdonik, S.B.: Approximate queries and representations for large data sequences. In: Proc. of ICDE, pp. 536–545 (1996)
7. Khalefa, M.E., Fischer, U., Pedersen, T.B.: Model-based Integration of Past & Future in TimeTravel (demo). In: PVLDB (2012)
8. Thomsen, C., Pedersen, T.B., Lehner, W.: RiTE: Providing On-Demand Data for Right-Time Data Warehousing. In: Proc. of ICDE, pp. 456–465 (2008)
9. Šikšnys, L., Khalefa, M.E., Pedersen, T.B.: Aggregating and disaggregating flexibility objects. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 379–396. Springer, Heidelberg (2012)
10. Garey, M.R., Johnson, D.S.: Computers and Intractability; A Guide to the Theory of NP-Completeness. W. H. Freeman & Co., New York (1990)

11. Boehm, M., Dannecker, L., Doms, A., Dovgan, E., Filipic, B., Fischer, U., Lehner, W., Pedersen, T.B., Pitarch, Y., Siksnys, L., Tusar, T.: Data management in the mirabel smart grid system. In: Proc. of EnDM (2012)
12. Taylor, J.W.: Triple Seasonal Methods for Short-Term Electricity Demand Forecasting. European Journal of Operational Research (2009)
13. Ramanathan, R., Engle, R., Granger, C.W.J., Vahid-Araghi, F., Brace, C.: Short-run Forecasts of Electricity Loads and Peaks. International Journal of Forecasting 13(2), 161–174 (1997)
14. Dannecker, L., Böhm, M., Lehner, W., Hackenbroich, G.: Forcasting evolving time series of energy demand and supply. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 302–315. Springer, Heidelberg (2011)
15. Dannecker, L., Schulze, R., Böhm, M., Lehner, W., Hackenbroich, G.: Context-aware parameter estimation for forecast models in the energy domain. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 491–508. Springer, Heidelberg (2011)
16. Dannecker, L., Böehm, M., Lehner, W., Hackenbroich, G.: Partitioning and multi-core parallelization of multi-equation forecast models. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 106–123. Springer, Heidelberg (2012)
17. Fischer, U., Boehm, M., Lehner, W.: Offline design tuning for hierarchies of forecast models. In: Proc. of BTW (2011)
18. Fischer, U., Böhm, M., Lehner, W., Pedersen, T.B.: Optimizing notifications of subscription-based forecast queries. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 449–466. Springer, Heidelberg (2012)
19. NREL, Wind Integration Datasets (2012), http://www.nrel.gov/wind/integrationdatasets/
20. JRC study on Smart Grid projects in Europe (2011), http://ses.jrc.ec.europa.eu/index.php?option=com_content&view=article&id=93&Itemid=137
21. Konsman, M., Rumph, F.-J.: D2.3. Final data model, specification of request and negotiation messages and contracts (2011), http://wwwdb.inf.tu-dresden.de/miracle/publications/D2.3.pdf
22. Verhoosel, J., Rothengatter, D., Rumph, F.-J., Konsman, M.: D7.5. MIRABEL-ONE: Initial draft of the MIRABEL Standard (2011), http://wwwdb.inf.tu-dresden.de/miracle/publications/D7.5.pdf
23. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: Proc. of SIGMOD (2006)
24. Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Erwin, C., Galvez, E., Hatoun, M., Hwang, J.H., Maskey, A., Rasin, A., Singer, A., Stonebraker, M., Tatbul, N., Xing, Y., Yan, R., Zdonik, S.: Aurora: A data stream management system. In: Proc. of SIGMOD (2003)
25. Kemper, A., Neumann, T.: Hyper: A hybrid oltp & olap main memory database system based on virtual memory snapshots. In: Proc. of ICDE (2011)
26. Thiele, M., Lehner, W.: Evaluation of load scheduling strategies for real-time data warehouse environments. In: Castellanos, M., Dayal, U., Miller, R.J. (eds.) BIRTE 2009. LNBIP, vol. 41, pp. 84–99. Springer, Heidelberg (2010)

Data Mining in Life Sciences: A Case Study on SAPs In-Memory Computing Engine

Joos-Hendrik Boese[1], Gennadi Rabinovitch[1], Matthias Steinbrecher[1], Miganoush Magarian[1], Massimiliano Marcon[1], Cafer Tosun[1], and Vishal Sikka[2]

[1] SAP Innovation Center, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
[2] SAP, 3410 Hillview Ave, Palo Alto, CA 94304, USA
<firstname>.<lastname>@sap.com

Abstract. While column-oriented in-memory databases have been primarily designed to support fast OLAP queries and business intelligence applications, their analytical performance makes them a promising platform for data mining tasks found in life sciences. One such system is the HANA database, SAP's in-memory data management solution. In this contribution, we show how HANA meets some inherent requirements of data mining in life sciences. Furthermore, we conducted a case study in the area of proteomics research. As part of this study, we implemented a proteomics analysis pipeline in HANA. We also implemented a flexible data analysis toolbox that can be used by life sciences researchers to easily design and evaluate their analysis models.

Topics: Data mining and data analysis in real-time, Case studies.

Submission type: Industry paper.

1 Introduction

The amount of data generated by life science research has been rapidly increasing over the last years. Examples include genome sequencing and mass spectrometry for proteomics. This data requires extensive and complex analysis, thus demanding high performance from data management systems. Today these analysis processes are typically implemented using heterogeneous systems, applications and scripts. The data sources are also diverse, with data distributed over different files or systems and being in different formats. Hence, supporting heterogeneity of application logic and data sources is a critical requirement on data management systems in this domain.

While Business Intelligence (BI) and other analytical enterprise systems have traditionally operated on pre-aggregated data, recent technologies are enabling analysis of highly structured multi-dimensional data directly on the raw data sources in real-time [1]. Moreover, modern analytical systems include interactive tools allowing the data scientists to explore data using relational operations and interact with the data at the "speed of thought". At the same time, these systems evolved from pure SQL query engines to multipurpose execution engines that can execute arbitrary logic directly on the data.

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 23–36, 2013.

Such a system is the SAP HANA in-memory database [2]. In contrast to traditional disk-based relational databases, SAP HANA keeps its primary data permanently in memory in order to achieve high analytical performance without pre-computed index structures. This enables fast execution of complex ad-hoc queries even on large datasets. Furthermore, HANA includes multiple data processing engines and languages for different application domains. These characteristics make HANA a promising platform to meet the data management requirements arising in life sciences.

In the scope of our life science projects, we work in close collaboration with research centers, universities and bioinformatic companies. Our goal is to provide the life science research community with fast and flexible data management tools based on HANA. In one of our project collaborations with the Freie Universität Berlin we concentrate on proteomics research. Proteomics is the branch of molecular biology that deals with the analysis of the tens of thousands of proteins present in a living organism. A very important application of proteomics research is the detection of biomarkers associated with different diseases including cancer. In order to detect these biomarkers, researchers must be able to flexibly analyze terabytes of data about the distribution of proteins in blood samples.

In this paper we present our case study in the field of proteomics and describe how SAP HANA in-memory database technology is used to support proteomics research. Moreover, we present a flexible and extendible framework for the real-time analysis of proteomics data that we developed based on HANA. We conclude that although HANA is initially designed for enterprise analytics, it is also well-suited for data management and analytics in life sciences.

The rest of this paper is structured as follows. In Section 2, we describe the SAP HANA in-memory database architecture and its distinctive features. A set of requirements for life science database systems that we identified during our collaboration with life science researchers is summarized in Section 3. In Section 4, we then provide a brief introduction to the field of proteomics, sketch the analysis pipeline used in this case study and describe the implementation of our HANA-based proteomics data analysis framework. Finally, Section 5 concludes the paper and presents some of our future work.

2 The SAP HANA Database Management System

The SAP HANA database management system [2,3] provides the data management infrastructure for SAP's future enterprise applications. The technical architecture and design of HANA was driven by the increased demand of modern enterprise applications for analytical performance as well as recent developments in hardware architectures. Requirements of modern enterprise applications include complex data mining and machine learning functionalities, increased data volumes and being able to answer ad-hoc queries on large datasets within seconds or subseconds.

In modern enterprise applications, data analysis tasks go beyond classical reporting such as OLAP queries and call for non-standard features like planning,

optimization, or predictive analysis features. Often, these tasks rely on non-relational data models, such as graphs or semi-structured data. To efficiently support these tasks, HANA keeps the primary copy of the data constantly in memory. Moreover, HANA supports a high-level of parallelism and cache-conscious execution of queries. This enables answering ad-hoc queries quickly and at any time without costly pre-computation of special index structures, as is usually done in traditional data warehouse systems. HANA follows a holistic approach to data management by consolidating transactional (OLTP) and analytical (OLAP) workloads in a single system.

To meet the requirements of a number of heterogeneous and complex data-intensive applications, HANA includes multiple storage and query engines:

Relational Engine: The majority of enterprise applications rely on relational data accessed via SQL. To support the standard SQL features, HANA provides a relational storage and query engine. Depending on the expected workload, the relational store can physically organize data column-wise or row-wise. Column-oriented data organization favors analytical processing of the data, since it allows cache-efficient processing of analytical operators and high compression rates [4]. In practice it turned out that scan operations on compressed columns are so fast that in most cases there is no need to create and maintain indexes, thus reducing the memory consumption.

Text Engine: To discover information hidden in unstructured text and combine it with structured information, HANA embeds a text engine that supports text indexing and a comprehensive set of text analysis features. Supported features range from various similarity measures to entity resolution capabilities and allow phrase or fuzzy search on text indexes.

Graph Engine: The graph engine provides access to data stored in graph structures, as commonly required by SAP's planning and supply chain applications. Domain-specific graph languages are supported to query and manipulate stored graph data.

Besides the ubiquitous SQL, HANA supports several domain-specific languages to process data managed by the different storage engines. These languages include: (i) MDX for multi-dimensional expressions, (ii) a proprietary language for planning applications called FOX, (iii) SQL extensions for text search and (iv) a proprietary language called WIPE to access and manipulate graph structures.

Functionality that cannot be expressed in SQL or one of the above languages can be implemented using a procedural extension to SQL called SQLScript. SQLScript adds imperative elements to SQL such as loops and conditionals allowing to define the control flow of applications. SQLScript can be coupled with operators defined by HANA's domain-specific languages. If performance is paramount, functionality can be directly implemented in C++ using native database interfaces within the database kernel. Since in this case the developer can implement parallel execution at a very low level, these native functions can be programmed with maximum control on parallelism during execution.

HANA ships with multiple preinstalled C++ libraries that implement frequently required functionality in enterprise applications. Examples of such

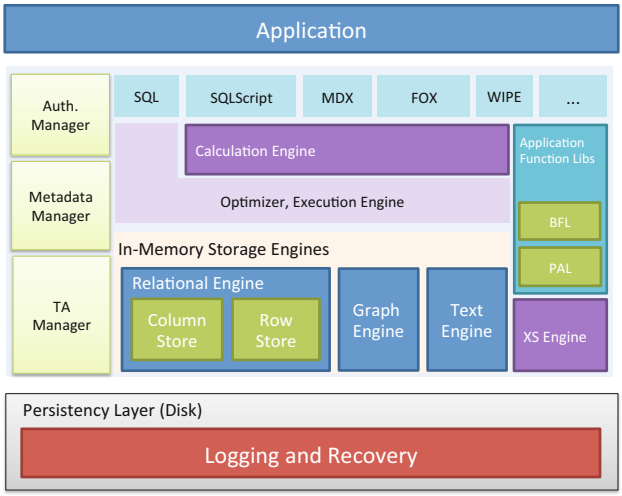


Fig. 1. Overview of the HANA architecture [2]

libraries include the business function library (BFL) and the predictive analytics library (PAL). The PAL implements standard machine learning algorithms, while the BFL provides functions frequently used in enterprise applications such as currency conversion etc.

Figure 1 depicts an architectural overview of HANA with its different languages, storage engines and libraries. In order to execute logic implemented in different domain-specific languages or available in function libraries, HANA generates complex programs called "calculation models" [5,3]. A calculation model defines the data-flow graph of a program executed by HANA's calculation engine. The edges of the data-flow graph connect data sources (i.e. tables, views, or results of other calculation models) and operator nodes. Operator nodes are functions implementing arbitrary logic defined in a domain-specific language and can have multiple input edges. Calculation models can also be exposed as database views and thus be easily reused as data sources for other calculation models or standard SQL queries. Current types of operator nodes in a calculation model are:

Relational Operator Nodes. All common relational operators (such as join, union, projection etc.) that handle classical relational query processing can be embedded.

R Operator Nodes. R nodes allow to embed logic defined in the R [6] language into the calculation model. In this case the data is copied into an R execution environment outside of the HANA database kernel. Results are then moved back into the calculation engine and integrated into the data flow of the calculation model.

L Operator Nodes. L is the internal runtime language of the HANA database and a subset of C++. It is used as an intermediate language into which all domain-specific languages are compiled. While currently end-users have no access to L, it can be used for internal development.

Custom Nodes. Custom nodes allow access to native functions implemented in C++ and are tightly coupled with the database kernel. Examples include functions defined in PAL or BFL.

Split and Merge Nodes. In addition to functional operators, split and merge operators are provided to define application specific data-parallelization. Split nodes partition the data for parallel processing, while merge nodes are used to combine results from different data partitions.

While SQLScript programs are transparently compiled into optimized calculation models, these models can also be defined explicitly using either an XML description or a graphical modeling tool, which is part of HANA Studio. HANA Studio is a client application for creating data models and stored procedures, or manage tables and issue queries interactively. It provides graphical editors for calculation models, thus allowing to easily define and manipulate analysis pipelines for enterprise applications as well as for life science research.

To conveniently manage the lifecycle of calculation models, HANA includes a built-in repository that is used to manage calculation models and other development artifacts, such as views and procedures. The built-in repository provides the basis for concepts like namespaces, versioning, and software component delivery.

Besides defining and managing calculation models, visualization and consumption of results is crucial. The Extended Scripting Engine (XS Engine) is a HANA component that supports consumption of results via an HTTP REST interface, e.g. for visualization purposes. Therefore, in most cases, developers do not need to integrate additional software components such as web servers or write application-specific code for data access. However, if the application requires some very specific mapping functionalities, these can be easily incorporated into the XS Engine in the form of additional software modules.

3 Requirements of Data Management in Life Sciences

In life sciences, the ability to manage and analyze large data sets is crucial. Data analysis in life sciences is often carried out through a diversity of scripts and applications developed in different languages. Data sources are also heterogeneous, with data spread across different systems and stored in different formats. This combination of heterogeneous applications and data sources requires costly data transfers and conversions, as well as the knowledge of different programming languages and models. Maintaining and processing all scientific data in a single centralized system would eliminate the need for expensive data transformation, thus enhancing performance and enabling researchers to concentrate on analysis algorithms, rather than on data transformation tasks.

The database community recognized the aforementioned problems and studies have been carried out to identify requirements for data management in life sciences [7].

Specialized scientific database systems, such as SciDB [8], have been developed to support life science applications. Other approaches like the one described in [9] focused on integrating data from multiple databases into a federated database system, allowing for a unified query interface for heterogenous life science data.

In the remainder of this section, we describe some of the important requirements we identified while supporting life science research groups, and explain how HANA addresses these requirements. Some of the identified requirements are in line with the general requirements for scientific databases mentioned in [7].

3.1 Tight Integration of Scientific Data and Analysis Algorithms

Consolidating different data sources and applications into a more homogeneous, centralized system removes the need for most data transfers and conversion across system boundaries. Nevertheless, data still needs to be transferred between the database's persistency layer and the application for computation, thus causing a significant performance loss. To reduce data movement between database and application, the database needs to be able to execute user-defined analysis algorithms directly on the data.

As mentioned in Section 2, HANA addresses the heterogeneous needs of complex enterprise applications by embedding multiple storage and query engines that support different domain-specific languages.

Furthermore, HANA is highly extensible and enables users to implement their domain-dependent application logic at various abstraction levels, ranging from high-level SQLScript procedures to native function libraries. This allows for more flexibility than traditional stored procedures or user-defined functions. As described in Section 2, application logic can be pushed down to be executed on primary copy of data which is held in memory resulting in high performance. Since all application logic is executed by the calculation engine directly on the data, there is no need for any data transfer beyond system boundaries.

3.2 Intuitive Interface for the Design of Analysis Pipelines

Life science researchers continuously design and test new analysis pipelines. This involves changing data sources, operations and parameters. Furthermore, researchers need to compare the results generated by different pipelines. Therefore, providing an easy-to-use and intuitive framework to interactively design analysis pipelines would greatly improve researchers' productivity.

In HANA, analysis pipelines are defined through calculation models, which can be designed in HANA Studio through a graphical interface. This interface can also be extended with domain specific operator nodes, e.g. custom nodes organized in toolboxes for different application domains.

3.3 Versioning of Algorithms and Data

Analyzing scientific data and comparing different data mining models is a complex task. Depending on some intermediate results, the requirements for further data evaluation and analysis may change over time. This often results in modifications of the algorithms or even whole data analysis processes. Discarded approaches, as well as all the intermediate results, are usually thrown away. If this intermediate data is needed at some later point, its reconstruction is often difficult and time-consuming. Versioning of algorithms and results helps addressing this problem. More specifically, in the case of a database system, both database objects (e.g. tables, stored procedures, user-defined functions etc.) and the stored data need to be put under version control.

Because every calculation model activated in HANA is stored and registered in a repository (see Section 2), it is easy to re-activate previously executed analysis steps. Also, because data in HANA is modified in a transactional context, every data record is associated with a transaction identifier. A mapping of transaction identifiers to revisions of calculation models can be used to easily identify the calculation model used to generate a specific record.

3.4 Support for Non-relational Data Structures and Operations

Processing of scientific data often requires domain-specific data structures and operations. While relational models are widely used to store scientific data and the associated metadata, emulating data structures such as graphs or hierarchies on top of pure relational tables results in poor performance. In order to achieve adequate performance, basic data structures such as graphs should be supported natively. HANA's holistic data management design follows this approach by integrating multiple in-memory storage engines for non-relational data such as graphs and including specific operators to access and manipulate non-relational data.

3.5 Big Data Support

Scientific datasets are steadily increasing in size, thus requiring tools to store, access and mine terabytes or petabytes of data. While large amounts of data must be stored and managed, the more complex and specific analysis tasks often touch only a fraction of this data. For example, data is often filtered or aggregated before an analysis model is run on the filtered data. For such filter and aggregation tasks Map-Reduce frameworks are frequently used. These frameworks scale transparently to large numbers of commodity machines, thus allowing to compute simple pre-processing, filter, and aggregation tasks in a massively parallel manner.

To exploit Map-Reduce implementations such as Hadoop for pre-processing or filtering of data sets that are too large to be loaded into HANA or that are already residing in a Hadoop cluster, some extensions need to be made to HANA. According to SAP's recently announced Unified Database Strategy [10],

HANA is going to support the access of such "big data" sources and offer a deeply integrated pre-processing architecture, allowing it to efficiently incorporate pre-processed data and to initiate post-hoc analyses with a Map-Reduce implementation.

4 Case Study: Proteomics Research

In this section we describe a proof-of-concept implementation that shows how HANA supports life sciences applications. Specifically, our contribution focuses on proteomics research. In Section 4.1, we first provide some general information about proteomics research and the analysis pipeline that we target in our HANA-based framework. In Section 4.2 we describe the implementation of this analysis pipeline in HANA.

4.1 Proteomics Research

Proteomics is the area of life sciences which focuses on the study of the proteome, a term referring to the totality of proteins expressed by a genome, cell, tissue or organism at a given time.

The human proteome is highly dynamic, changing constantly as a response to the needs of the organism. Depending on factors such as sex, age, diet and stress, the proteome differs widely between different individuals. Other factors, like cancer or other diseases, can also change the composition of the proteome. These changes in the proteome are indicated by the presence or absence of certain proteins or protein combinations. An important application of proteomics is to analyze the proteome of different groups of people, i.e. male/female or healthy/cancer patients, and to search for patterns within these groups that clearly assign a single patient to the appropriate group. These patterns can then be used as biomarkers for the respective group, i.e. in cancer diagnosis.

A prominent way of analyzing the proteome is to use mass spectrometry, a technique that measures the masses and concentrations of proteins in a biological sample and exports the data in form of a spectrum. An example of mass spectrum is shown in Figure 2. In our case study, blood samples are analyzed using the MALDI-TOF[1] mass spectrometry technique which is illustrated and described in Figure 3. The process is carried out multiple times per sample to increase the coverage, leading to a multitude of spectra per blood sample.

Spectrum Data. The analysis of a single blood sample results in 1,500 to 2,000 spectra with approximately 100,000 data points (mass/count pairs) each. Thus, each sample results in 150-200 million data points. Because these data points are generated by external tools, they are usually provided in CSV[2] or specific mass spectroscopy data formats such as mzXML[3]. The raw data we use for our

[1] Matrix-assisted Laser Desorption/Ionization, Time-of-Flight mass spectrometry.
[2] Comma-separated values.
[3] `http://tools.proteomecenter.org/wiki/index.php?title=Formats:mzXML`

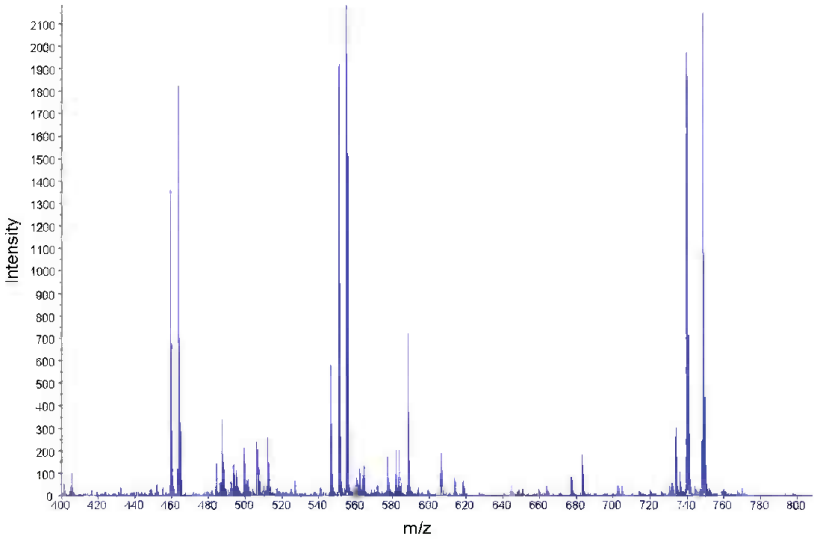


Fig. 2. Portion of a raw protein mass spectrum showing pronounced peaks in the concentration (y-axis) of proteins corresponding to certain mass-to-charge ratios (x-axis). Peaks in the concentration of certain proteins may indicate the presence of a disease, e.g. cancer.

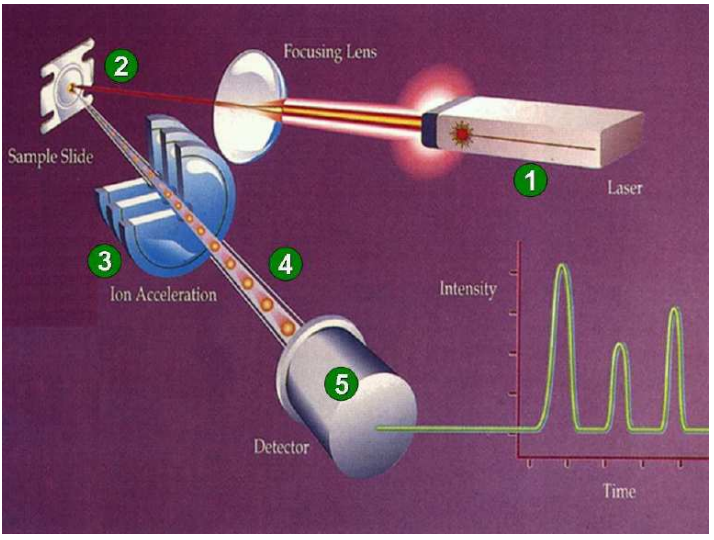


Fig. 3. Operation of a modern MALDI-TOF mass spectrometer. A laser beam (1) vaporizes and charges the molecules in the sample (2). These are accelerated in an electrical field (3) and their time of flight in the drift region (4) is measured by a detector (5). (picture taken from [11], therein modified and cited from [12]).

case study amounts to two gigabytes per sample. When this data is imported into the HANA database, in-memory compression reduces its size to about one gigabyte. More details on the compression ratio are provided in Section 4.2.

Analysis Pipeline. In this study, we consider the analysis pipeline proposed in [11]. This pipeline takes as input raw mass spectra and produces a predictive model that can be used to classify patients across relevant groups, such as healthy or diseased. The pipeline consists of four major steps:

1. **Preprocessing**
   Removal of noise and other systematic errors from the mass spectrum.
2. **Peak-Seeding**
   Identify peaks within the preprocessed spectra. This includes the identification of potential peaks and the derivation of features describing each peak.
3. **Peak-Picking**
   Examine the peaks across a group of samples (e.g. healthy patients) and group similar peaks occurring across the samples into clusters relevant for further analysis. These clusters of peaks are called master peaks.
4. **Analysis**
   Find master peaks that are able to separate the group of healthy patients from the group of diseased and therefore contribute to a corresponding biomarker.

The peak-seeding phase finds relevant peaks in a single raw spectrum. Each peak in the spectrum is described by the parameters of a Gaussian function which is fitted to the raw data points contributing to this peak. This process also includes deconvolution of overlapping peaks. The result of the peak-seeding phase is a set of peaks from all spectra of all samples, where each peak is described by a distribution and additional parameters such as start mass, end mass, contributing points, raw height and fit quality.

In the peak-picking phase, for each group of samples (e.g. healthy patients) similar peaks occurring across the samples of the group are then grouped into clusters of peaks. A cluster of similar peaks defines a so-called master peak.

Finally, the set of master peaks of two different groups (e.g. healthy/diseased) is analyzed to determine a subset of master peaks that is successfully able to distinguish between the two groups. This subset of master peaks can then be used to classify new samples and is the base for developing novel medical tests e.g. for cancer diagnosis.

As the data flows through the analysis pipeline, its size diminishes: The raw unprocessed spectrum in step 1 constitutes the largest data volume. The output of the peak-seeding phase are only those parts of the spectrum that represent actual peaks, thus introducing some degree of semantic meaning and reducing the data (10%–20% less than the raw spectrum size). The remaining peak-picking and analysis steps further reduce the size of data passed on in the analysis pipeline.

The reduction of data within each processing step depends on the resolution of peak-seeding and picking, i.e. whether small peaks are either considered noise or are passed on to subsequent analysis steps. Since master peaks with a relatively small height in the raw data might make the difference in identifying relevant biomarkers, researchers change thresholds for peaks to be considered in peak-picking and analysis. Identifying biomarkers is, therefore, an interactive process that requires re-running of the analysis pipeline. Thus, fast and predictable responses are crucial to support the workflow of proteomics researchers operating this pipeline.

4.2 Proteomics on HANA

We implemented the proteomics analysis pipeline described in Section 4.1 in the HANA database using multiple calculation models that together define the proteomics analysis pipeline. Moreover, we provide a graphical editor that enables interactive execution and manipulation of the analysis pipeline. Besides integration of proteomics-specific operators, the editor also enables an easy integration of operators from the PAL.

Implementation of the Proteomics Analysis Pipeline. Data received from the MALDI-TOF mass-spectrometer is pre-processed outside of HANA to remove systematic errors and noise. The pre-processing step is executed outside of HANA because it requires only one pass through the data and can thus be easily performed while sequentially reading the raw data from files. The resulting pre-processed spectra are then loaded into HANA's column-store in a single import operation. The product of the import operation is a single column table containing the pre-processed mass-spectra for all samples.

Storing the spectrum data in a column table allows for compression of the data at a ratio of approximately 1:2. Although a mass spectrum contains mainly double and integer values, the main memory required to store a single sample is about 1.4 GB compared to 2.4 GB required in a CSV file. Studies in proteomics research rarely consider more than 1,000 samples due to expensive data acquisition, so we assume that a HANA appliance with 2 TB of main memory has sufficient capacity for most studies. In case the number of samples exceeds the available main memory, HANA can be scaled out across multiple machines.

The peak-seeding phase requires the determination of Gaussian components using standard minimization methods. Since minimization is an iterative process, this algorithm was implemented in C++ within the database kernel for a maximal performance. The peak-seeding function operates on single spectra of a sample and is, therefore, easily parallelizable. Since the peak-seeding algorithm operates on intensity values of specific mass ranges in a spectra, the column-oriented organization of spectra data is beneficial, as the intensities of a mass range can be scanned efficiently.

The peak-picking step is not implemented in C++, but in L and SQLScript since the data sizes the peak-picking step operates on are much smaller than the

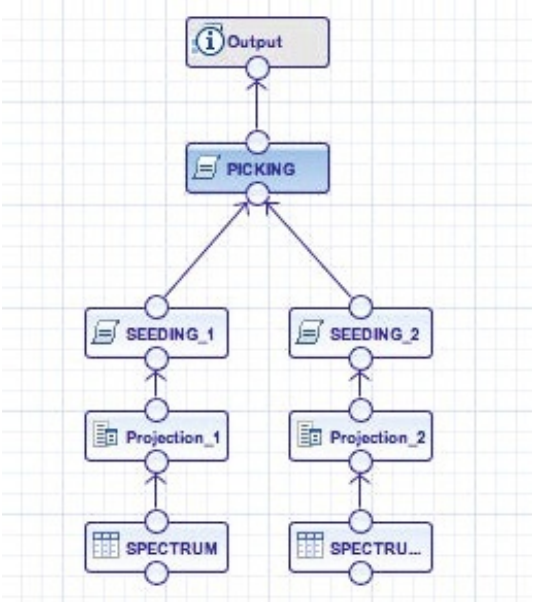


Fig. 4. A fragment of the proteomics analysis pipeline, that includes peak-seeding and peak-picking steps

spectrum data processed during the peak-seeding step. The peak-picking step also makes use of functions implemented in the PAL.

Subsequent analysis steps of the pipeline aggregate master peak data further, e.g. by computing statistical measures of master peaks and comparing peaks between two sample groups (e.g. between healthy and diseased patients) to select appropriate biomarkers. These steps work on master peak data and can, therefore, be expressed efficiently in SQL, SQLScript, and PAL operators.

Calculation Model Editor. We used the calculation model described in Section 2 to implement the proteomics analysis pipeline. To design the calculation model, we extended the HANA Studio modeling tool with a data-mining plugin that enables integration of domain-specific L and Custom operator nodes. For our case study, we implemented a domain specific library for proteomics research. In addition to domain-specific operator nodes, the data-mining plugin contains nodes for standard machine learning algorithms, such as k-means and C.4.5, which are implemented in the PAL. The user can simply drag and drop operator nodes and orchestrate them in the data-flow graph of the calculation models.

Figure 4 depicts a calculation model created in HANA Studio using our data-mining plugin. The depicted model is part of the analysis pipeline that includes peak-seeding and peak-picking steps. The peak-seeding and peak-picking operator nodes are provided by our proteomics library. The data sources are database tables with sample spectrum data. To implement the peak-seeding step, we

connect each data source to a peak-seeding operation node to detect the peaks within the spectra of the data source. We also interpose a projection operator between the data source and the peak-seeding step with the purpose of mapping data source attributes to the signature of the operator node. Finally, peak-picking is performed by grouping the peaks identified in both spectra sources during the peak-seeding phase. The final results are provided by an output operator that projects the relevant attributes of the table containing the results of the peak-picking phase. The results can then be used as a data source in other calculation models.

After designing the analysis pipeline, the model can be activated and executed on the HANA calculation engine and results can be previewed in the form of diagrams and tables. Therefore, the tool constitutes a flexible design interface where the user can customize and adjust the pipeline steps, e.g. peak-picking, by providing different values for input parameters or by easily change the operators in the calculation model. In this way, the user can modify and re-run analysis pipelines and conveniently compare their results.

5 Summary and Conclusion

In this paper, we identified a set of requirements for data analysis and management in life sciences research and described how HANA meets these requirements. We made the point that data management tools like HANA, although initially designed for enterprise analytics and BI, are also a good fit for data management in life sciences.

As a concrete example of a life sciences application on top of HANA, we presented a proof-of-concept implementation of a proteomics analysis pipeline within the HANA database kernel and described how different analyses were implemented and how the pipeline was orchestrated in calculation models. The concept of calculation models in HANA and the high analytical speed allow life sciences researchers to interactively analyze data using graphical modeling tools and to easily design and modify their analysis tasks. We showed how an extension to the HANA Studio can be used for such explorative domain-specific data analysis.

Although our work is a promising first step, much remains to be done in order to successfully use in-memory databases for life sciences. While we thus far focused on flexibility and extendibility, our next goals are visualization of results and performance evaluation. With respect to visualization, we plan to extend our framework with interactive visualization components that allow users to easily inspect the analysis results. Moreover, we plan to make it easy for applications to consume the results, by exposing them through standard HTTP using HANA built-in XS Engine. With respect to performance, we plan to benchmark our solution against DBMSs currently used for life sciences research.

References

1. Plattner, H., Zeier, A.: In-Memory Data Management: An Inflection Point for Enterprise Applications. Springer (June 2011)
2. Färber, F., Cha, S.K., Primsch, J., Bornhövd, C., Sigg, S., Lehner, W.: SAP HANA Database: Data Management for Modern Business Applications. SIGMOD Rec. 40(4), 45–51 (2012)
3. Sikka, V., Färber, F., Lehner, W., Cha, S.K., Peh, T., Bornhövd, C.: Efficient Transaction Processing in SAP HANA Database: The End of a Column Store Myth. In: Proceedings of the 2012 International Conference on Management of Data, SIGMOD 2012, pp. 731–742. ACM, New York (2012)
4. Plattner, H.: A Common Database Approach for OLTP and OLAP using an In-Memory Column Database. In: Proceedings of the 35th SIGMOD International Conference on Management of Data, SIGMOD 2009, pp. 1–2. ACM (2009)
5. Jaecksch, B., Faerber, F., Rosenthal, F., Lehner, W.: Hybrid Data-Flow Graphs for Procedural Domain-Specific Query Languages. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 577–578. Springer, Heidelberg (2011)
6. Venables, W.N., Smith, D.M.: An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics Version 2.15.0. R-project.org. (March 2012)
7. Stonebraker, M., Becla, J., DeWitt, D.J., Lim, K.T., Maier, D., Ratzesberger, O., Zdonik, S.B.: Requirements for Science Data Bases and SciDB. In: CIDR (2009)
8. SciDB.org: The Open Source Data Management and Analytics Software for Scientific Research, `http://www.scidb.org/Documents/SciDB-Summary.pdf` (last accessed: July 22, 2012)
9. Haas, L.M., Schwarz, P.M., Kodali, P., Kotlar, E., Rice, J.E., Swope, W.C.: DiscoveryLink: A System for integrated Access to Life Sciences Data Sources. IBM Syst. J. 40(2), 489–511 (2001)
10. SAP: SAP Unveils Unified Strategy for Real-Time Data Management to Grow Database Market Leadership. SAP News (April 2012), `http://www.sap.com/corporate-en/press.epx?PressID=18621`
11. Conrad, T.O.F.: New Statistical Algorithms for the Analysis of Mass Spectrometry Time-Of-Flight Mass Data with Applications in Clinical Diagnostics. PhD thesis, Freie Universität Berlin (2008)
12. MALDI-TOF Mass Analysis, `http://www.protein.iastate.edu/maldi.html` (last accessed: July 25, 2012)

The Vivification Problem in Real-Time Business Intelligence: A Vision

Patricia C. Arocena, Renée J. Miller, and John Mylopoulos

University of Toronto, Toronto M5S 3G4, Canada
{prg,miller,jm}@cs.toronto.edu

Abstract. In the new era of Business Intelligence (BI) technology, transforming massive amounts of data into high-quality business information is essential. To achieve this, two non-overlapping worlds need to be aligned: the Information Technology (IT) world, represented by an organization's operational data sources and the technologies that manage them (data warehouses, schemas, queries, ...), and the business world, portrayed by business plans, strategies and goals that an organization aspires to fulfill. Alignment in this context means mapping business queries into BI queries, and interpreting the data retrieved from the BI query in business terms. We call the creation of this interpretation the vivification problem. The main thesis of this position paper is that solutions to the vivification problem should be based on a formal framework that explicates assumptions and the other ingredients (schemas, queries, etc.) that affect it. Also, that there should be a correctness condition that explicates when a response to a business schema query is correct. The paper defines the vivification problem in detail and sketches approaches towards a solution.

Keywords: data exchange, vivification, incompleteness, uncertainty, business intelligence.

1 Introduction

Every time a Business Intelligence (BI) query is evaluated, the data that are returned need to be *interpreted*. Interpretation involves mapping data to business entities, processes and goals that correspond to BI data. For example, a hospital database may contain data about patient names, addresses and health insurance numbers (hi#). Interpretation of these data means that some of them (e.g., name = 'John Smith', address = '40 St. George Street', hi# = 1234567) are ascribed to one patient. Likewise, the database may contain data about events, a hospital admission, an assignment to a ward, and a surgical operation. Interpretation, in this case, means that these events are associated with the entities that participated in them. Moreover, these events are grouped into aggregates, where each represents an instance of a business process. For example, an aggregate such as 'HospitalVisit' models the day-to-day activities in a hospital. This form of interpretation has been called *vivification*, in the sense that it brings data to life (hence, vivifies), or makes data more vivid [13].

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 37–49, 2013.

The past decade has seen unprecedented interest in BI technologies and services, and a corresponding growth of the BI market. By now, most competitive organizations have a significant investment in BI, much of it technology-related, based on software tools and artefacts. But business people - be they executives, managers, consultants, or analysts - are in general agreement that what helps them the most is business data analyzed in business terms: strategic objectives, business models and strategies, business processes, markets, trends and risks. This gap between the world of business and the world of IT-supplied data remains today the greatest barrier to the adoption of BI technologies, as well as the greatest cost factor in their application to specific projects.

Today's business world requirements for information-on-demand and agility exacerbate this gap. Information-on-demand is hampered when BI tools handle data integration and cleansing of IT-supplied data in isolation from any form of business analysis and interpretation. Pressed with increasing volumes of data and deployment times, this approach usually results in a tendency to integrate and clean as much data as possible with very little business perspective in mind. During the consolidation phase, BI systems (and thus the technologies managing data warehouses over which BI applications are built) make vivification choices when confronted with *incomplete* and *inconsistent* information. But as data and business semantics continuously evolve, *one-size-fits-all* interpretations of the data are rarely sufficient for business analysts. In practice, the evaluation of business queries and interpretation of results amount to a continuous social discovery process between IT and business users, where the latter 'fill in the blanks' and routinely introduce assumptions to obtain the desired level and type of data completeness for decision making.

Agility means that *real-time change* is inevitable and ever-present in a business context, and must be factored into the interpretation and decision making processes as much as in everything else. Real-time requires the ability to access information in an organization whenever it is required [6]. In a real-time enterprise, business policies, objectives and processes change, obstacles and opportunities are encountered and must be dealt in a timely fashion. In our work, we seek solutions to enable BI systems to be able to react to these changes and graciously adapt any interpretation choices made so far without further need of re-architecting the data warehouse or redesigning ETL scripts. This adaptation must be performed at the same time scale as changes in semantics. So when a new business policy is invoked or a law changed, we must be able to adapt the interpretation in a timely fashion.

To bridge the gap between the business and IT worlds, we propose to use a *business model* (also known as business schema) to represent business concepts (e.g., business goals, entities and processes [12]) and, moreover, *schema mappings* from a database schema to a business model to specify interpretations. Data are then interpreted in terms of these concepts. Unfortunately (but unsurprisingly!), the vivification of data in terms of business model concepts is not a simple one-to-one mapping. Much like the task of finding who visited a crime scene on the basis of footprints, there are in general many possible mappings from data to

business entities and processes. For instance, the hospital database may mention N times patients named 'John Smith'. These may correspond to 1, 2, ... or N different patients. Likewise, a surgical operation may be part of one patient's hospital visit or another's, depending on the data associated with the operation, and how these data have been interpreted.

In this position paper, we elaborate on the need for a systematization of the vivification process, to make vivification rigorous and traceable. Specifically, we argue that solutions to the vivification problem should be based on a formal framework that explicates assumptions, records context, provenance and the other ingredients that entail an interpretation. Moreover, vivification should come with a correctness condition for a response to a business schema query to be correct. In other words, we call for vivification to be practiced with the same rigour that has been applied to query processing for more than three decades.

The rest of the paper is structured as follows. In Section 2, we discuss our research baseline. In Section 3, we consider how vivification can be used in BI, that is, we outline elements of a formal framework to bring data to life. In Sections 4 and 5, we present some steps involved in our envisioned vivification process and we illustrate using preliminary results from our case study. In Section 6, we summarize related work. Last, we conclude in Section 7.

2 Research Baseline

A business schema is most useful if it is instantiated with complete and accurate information. This information is naturally derived from the IT-supplied databases (or warehouses), but since these databases are often designed for a different purpose (supporting operational systems or supporting BI reporting), it is rare that they provide complete and consistent business data. Hence, business users must deal with missing and uncertain information on a daily basis. Next, we review two foundational techniques for interpreting data.

2.1 Knowledge Base Vivification

Consider a knowledge base (KB) as a finite collection of facts about the world of interest. We say a KB is *incomplete* when it asserts that one of many facts is true, but without being able to individually determine the truth of each individual fact. A KB indicating that 'John Smith' as a recently admitted inpatient is either in the Cardiology Ward or in the Maternity Unit, without saying which of the two locations holds, is considered incomplete. Incompleteness in this example arises due to a disjunction in the representation of the facts. In general, incompleteness may be attributable to a number of logical constructs, such as disjunction, existential quantification and negation [13].

To act in the face of incompleteness, Levesque proposed a technique called *vivification* [13] for transforming an incomplete KB into a complete representation, where reasoning can be reduced to standard database querying. Vivifying a KB amounts to eliminating incompleteness, and most importantly, placing symbols in the KB into a *one-to-one correspondence* to objects of interest in the

world and their relationships. For example, confronted with the disjunctive fact that 'John Smith' is either in the Cardiology Ward or in the Maternity Unit, we might be able to resolve this issue by looking up the location of the admitting physician's office. In doing so, we may use the relationship between admitted inpatients and admitting doctors in the business to establish proper connections among the data.

The notion of vivification is relevant to the task of interpreting the world of IT-supplied data in terms of the world of business. First, it corresponds well to the common *ad hoc* practice of 'filling in the blanks' to coerce data into a form that is complete and adequate for business querying. Second, various types of business data, such as business plans, strategies and goals, are already conveyed using vivid or visual descriptions. This is, ultimately, the type of data business users have in their mind when issuing queries. We propose to use vivification to provide a principled foundation for doing the data-to-business interpretation.

2.2 Data Exchange

Vivification is also important in translating between two databases [17]. Schema mappings are used to express the relationship between two database schemas. Given a schema mapping, the problem of data exchange was introduced by Fagin et al. [11] and defined as the problem of translating data structured under a source schema into an instance of a target schema that reflects the source data as accurately as possible. An important aspect of this work was the recognition that even in exchanging data between two IT database schemas, it is rare that a complete target instance (database) can be created. The notion of a *universal solution* was introduced as the best solution for representing data as accurately as possible, but these instances are often *incomplete* as we exemplify next.

Example 1. Consider a simple schema mapping stating how to populate a target insurance database from a source inpatient registry table:

$$\texttt{Registry}\,(name, dob, addr) \rightarrow \exists\, \textbf{policyId}\ \texttt{Insurance}\,(\textbf{policyId}, name, dob)$$

We use source-to-target tuple-generating dependencies (s-t tgds) to express the mapping [11]. This mapping indicates that for each inpatient registry tuple, there must be an associated insurance record in the target, whose name and date of birth values are the same as those of the inpatient registry. The variables $name, dob, addr$ are universally quantified. Notice that the target expression (right-hand-side of the implication) uses an existential variable `policyId` to indicate that the policy number is underspecified. Figure 1 shows a source instance I and a universal solution (target instance) J with labeled `NULL` values (i.e., N_1, N_2 and N_3) instead of concrete policy numbers.

Schema mappings record the decisions made in translating data. Data exchange permits the precise modeling of the uncertainty in a translation. In the above target instance, the use of three labeled `NULL`s indicates that the three tuples could refer to three different people, or to just two people (perhaps Marge

moved so the first two NULLs should share a single value) or to a single person (perhaps there were inconsistencies in entering information and all three tuples refer to one person). If a business needs to track the number of patients, someone must make a choice on how to resolve this uncertainty. We propose to build on the foundation of data exchange to view the vivification task as one of removing incompleteness or uncertainty from a schema mapping. We advocate a declarative approach where vivification decisions are documented in a formalism that supports automated reasoning (in the same way schema mappings permit formal reasoning about a translation) and in which we can define precisely when an interpretation (or set of vivification decisions) is vivid.

Source Instance I

Name	DOB	Address
Marge Wood	05-05-1927	Markham
Marge Wood	05-05-1927	London
Margret Wood	05-05-1925	London

Target Instance J

PolicyId	Name	DOB
N_1	Marge Wood	05-05-1927
N_2	Marge Wood	05-05-1927
N_3	Margret Wood	05-05-1925

Fig. 1. A Data Exchange Scenario: Source Instance and Universal Solution

3 Formal Framework

We now present elements of a formal framework for interpreting data in BI, using the concepts of database schema, business schema, business mapping, vivification assumptions and business queries.

Database Schema. A source database schema comprises a number of relations $R_1, R_2, \ldots, R_n$, some of which may be recording information about the instrumentation of business processes. Our proposal is general enough to accommodate not only mainstream relational databases but also specific technologies such as data warehouses.

Business Schema. A target business schema, on the other side, offers a conceptual model of a particular organization in terms of business concepts $T_1, T_2, \ldots, T_m$, such as business entities, business processes, and business events. This description may be given in some business modeling formalism [12].

Business Mapping. To describe the relationship between the source and the target schemas, we use a set $\mathcal{M}$ of *business schema mapping rules*. Some mapping rules indicate a correspondence between database tuples and business entities, and some others between observed events in the data and business events in the process model. The mapping language we use for illustration is based on the widely used database concept of s-t tgds [11]. Each mapping rule may indicate a correspondence between a query over the source database ϕ_{DB} and a query over the target ψ_{BS} involving either business entities or business events. Formally, each mapping rule is of the form:

$$\forall \mathbf{z}, \mathbf{x}(\phi_{DB}(\mathbf{z}, \mathbf{x}) \rightarrow \exists \mathbf{y} \psi_{BS}(\mathbf{x}, \mathbf{y})) \quad (1)$$

Intuitively, $\mathbf{x}$ represents the information that is exchanged from the source into the target, and $\mathbf{y}$ represents information that is unknown in the target.

Assumptions. We also use a set $\mathcal{A}$ of *vivification assumptions* to characterize desired properties or constraints over any instantiation of the target business schema. These may be given using any constraint language, such as tuple-generating dependencies (tgds) and equality-generating dependencies (egds) [2]. In addition to having functional dependencies and cardinality constraints, we may also have assumptions that state a particular choice among a set of possible facts to resolve incompleteness or inconsistency.

Business Queries. Last, we have a set $\mathcal{Q}$ of *business queries* in some query language formalism. We assume a traditional query answering semantics over complete databases which will determine for a given query what parts of a given business schema instance must be complete to compute a certain query answer. For example, if we consider Example 1, using this mapping alone, we would not be able to answer a query that counts the number of insurance policies. This mapping leaves the insurance policy identifiers incomplete, and therefore leaves incomplete the decision on whether two policies are the same. However, a query that asks if Marge Wood is a patient can be answered with certainty. We can use certain query answering semantics [3] to reason about whether an instance of a business schema is sufficient to answer with certainty a given set of queries.

Definition 1. *A BI setting* $(\mathtt{DB}, \mathtt{BS}, \mathcal{M}, \mathcal{A}, \mathcal{Q})$ *consists of a database schema* $\mathtt{DB}$*, a business schema* $\mathtt{BS}$*, a set* $\mathcal{M}$ *of business mapping rules, a set* $\mathcal{A}$ *of vivification assumptions, and a set* $\mathcal{Q}$ *of business queries. Let* I *be a source instance over* $\mathtt{DB}$ *and* J *a target instance over* $\mathtt{BS}$ *such that* $\langle I, J\rangle$ *satisfy* $\mathcal{M}$ *and* J *satisfies* $\mathcal{A}$*.*

- *We say that* J *is* vivid *if any tuple in* J *is null-free and all constants in* J *are justified by either* I *(the source instance at hand) or by* $\mathcal{A}$*.*
- *We say that* J *is* vivid for a set of queries $\mathcal{Q}$ *if for every query* $q \in \mathcal{Q}$*, the sets of possible and certain answers for* $q(J)$ *coincide.*

It follows from this definition that a vivid business schema instance needs to be *discovered*. This needs to be done in the light of concepts in the business schema, any given vivification assumptions, and a set of business queries of interest. To do this, our approach is to develop a refinement process for schema mappings that step-by-step leads to more vivid target instances. We use the business queries to understand what vivification is necessary to ensure business users receive certain answers that are based on documented assumptions.

A First Solution. We now provide a brief overview of our initial solution to the vivification problem. In Arocena et al. [5], we discuss specific strategies for vivifying underspecified attribute values arising from existential quantification in business mapping rules. In particular, as illustrated in Figure 2 Part (a), we consider how to remove *incompleteness* and complement this with a consideration of how to remove *uncertainty* that arises due to inconsistency. Incompleteness

may arise when a mapping rule fails to specify what value should be given to a target attribute or when the database I fails to provide a value, thus rendering an incomplete NULL value during mapping execution. Uncertainty may arise due to having more than one way of populating a given target relation or due to a database that does not enforce business-level requirements (for example, a business rule saying every patient must provide a single health insurance policy and the database provides multiple ones).

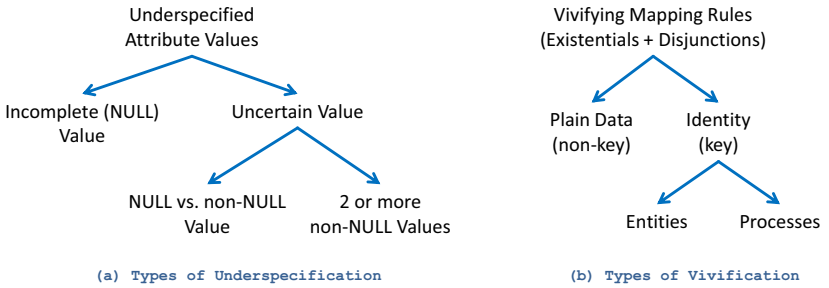


Fig. 2. Types of Underspecification and Vivification in Business Mapping Rules

Our approach uses formal reasoning about a business mapping and a set of assumptions to determine when they are sufficient to produce a vivid instance or an instance that is vivid with respect to a set of queries. To do this, we first consider the attribute positions at which incompleteness or uncertainty occurs in mapping rules, and then we consider the set of business queries and their requirements. In Example 1, for a query that counts patients, we need to make a decision (which we record in our vivification assumptions $\mathcal{A}$) on whether NULL values over the target attribute represent the same or different values), but we do not need to map each patient to their policy number. Of course, if such a mapping can be found, this is a valid way to vivify the incompleteness caused by the existential variable in the mapping rule. In general, when applying *attribute-level* vivification strategies, we distinguish between non-key and key attribute positions (shown in Figure 2 Part (b)). We refer to these as *data vivification* and *identity vivification*, respectively. The first is concerned with supplying one or more certain data values to a target business concept. The second one attempts to bring operational data to life in terms of business entities and processes by correctly associating data with specific instances of real-world business concepts. In the next section, we overview our proposed process.

4 Towards an Iterative Vivification Process

Figure 3 illustrates our proposed approach to vivification. We envision a process where steps can be iteratively applied as real-time information becomes available or changes and, moreover, as technologists and business users further their own understanding and knowledge of the deployed business solution.

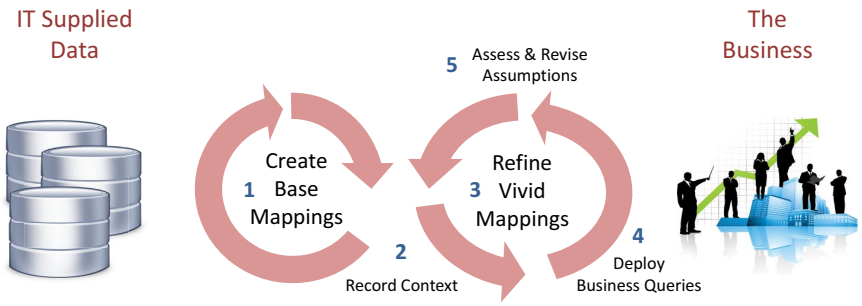


Fig. 3. An Iterative Vivification Process

Step 1: Create Base Mappings. Our vivification process starts by considering a *source database schema* and *target business schema*. The database and business schemas can be very different as they are often developed and evolve independently. The output of this step is an initial set of *business mapping rules* specifying the relationship between concepts in these two disparate schemas, as currently understood by users. These mapping rules may be derived using semi-automatic schema mapping systems, such as Clio [10] and ++Spicy [15], or extensions of these approaches designed for mapping into a conceptual schema [4].

Step 2: Record Context. Before attempting to map any data, we look to the business schema, the business process model, and importantly, to business queries to understand implicit contextual assumptions. In establishing a proper *context of interpretation*, we may also consider user-defined assumptions. This step helps to identify which incomplete or uncertain attribute values need to be vivified to instantiate the business schema with complete high-quality data. Recording the context of interpretation is crucial for understanding what types of vivification strategies may be applied and, later on, for understanding the rationale behind query responses.

Step 3: Refine Vivid Mappings. In this step, we use business context (i.e., the context of interpretation created in Step 2) to guide mapping vivification. We view this task as one of removing incompleteness or uncertainty from business mapping rules. This involves systematically choosing and applying vivification strategies. These include finding mapping tables or finding default values from the context to resolve existential variables and using business rules to resolve inconsistencies [5]. The output (after iteration) is a *refined vivified business mapping*, which chooses among all possible interpretations of the data, one that is consistent with the context of interpretation. We also record the *provenance* of these decisions.

Step 4: Deploy Queries. We use the refined vivified mapping of Step 3 to instantiate the target business schema with complete high-quality data. Once this happens, it can be queried by business users. Notably, most *business queries*

require the computation of information across business concepts. This computation often relies on *grouping* instances of a given business concept by common attributes (e.g., date of occurrence, location, participating entities, etc.) and, optionally, performing *aggregation* of these or some other attributes of interest to create business relevant performance indicators. Grouping and aggregation place strong requirements on the completeness of data when the ultimate goal is to obtain query responses that enable effective decision making.

Step 5: Assess and Revise Assumptions. A decisive step in our vivification process is that of assessing and revising the *assumptions* used during the interpretation of data. A new business mandate, a change in business process models, or even an improved business user's understanding of the application domain may all cause certain assumptions to not hold true. In such cases, we need to revise our assumptions and possibly *retract* any vivification done based on changed or deleted assumptions. Real-time change is ever-present in a business context and must be dealt with to ensure up-to-date consistency and completeness guarantees, as required by the business queries.

5 Validation: A Prospective Case Study

To validate our proposal, we are conducting a case study which considers the flow of patients attending an Emergency Department (ED) in a health care organization in Ontario. In this section, we briefly overview our initial undertaking.

Various health organizations in Ontario are deploying the Daily Activity Report Tool (DART), a performance management toolkit that monitors improvements to patient flow within hospitals. DART collects daily Emergency Department (ED) and General Internal Medicine (GIM) activity data to help illustrate a hospital's patient flow from start to end. In our work, we consider a *target business schema* outlining a basic *ED process model*, based on the schema developed by Barone et al. [7]. A small portion of our schema is depicted in Figure 4. The data collected by DART (described as 'Output Data' in the figure) can be seen as an instantiation of the process model. From this data, DART computes a set of mandatory *business queries* (also known as indicators) that can be used to track hospital performance, such as 'Time to Physician Initial Assessment (PIA)' and 'ED Length of Stay (LOS)' indicating the length of a patient's stay in the ED during one episode of care. The aim of DART is to empower hospital stakeholders in their daily discussions regarding patient flow.

Interpretive Challenges. We use a *source database schema*, which models operational hospital data, based on that of a leading hospital in Ontario. Most relational tables record information about patients and observed events with respect to activities that occur on or on behalf of patients. Interestingly, the concept of patient as an entity does not exist in the database. Instead, the recording of information hinges upon the concept of ED Visit (or episode of care), and various location-based, status and census data items. Moreover, across hospital sites, the hospital may use different chart or medical record identifiers

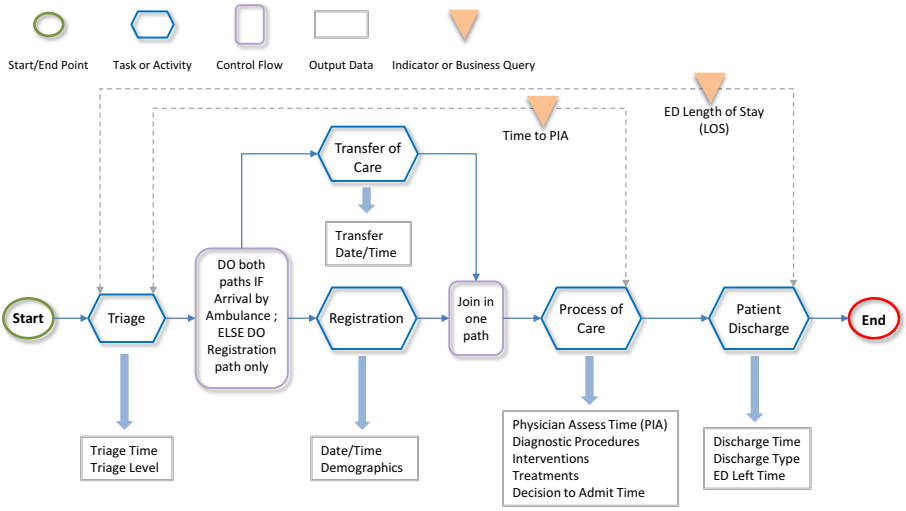


Fig. 4. Emergency Department Patient Workflow

to refer to a person's involvement in an episode of care. As a result, counting ED patients at any given point in time is a challenge (surprisingly, this count happen to be one of the most widely used indicators within DART). Another challenge encountered in our application domain is that of missing and uncertain information. Missing values mostly occur over the following business schema target attributes: Triage Acuity Level, Triage Time, Physician Assessment Time (PIA), Physician Identifier and ED Departure Time. In some cases, it has been estimated that close to 25% of ED visits may have missing data [1]. For example, census times are mostly recorded manually and thus, when not recorded, they may impact the interpretation of business indicators.

Our Approach. As we proposed in Section 4, we are deploying an iterative vivification process over our case study. We briefly illustrate the use of our techniques with two examples.

Example 2. A relevant business query in our case study is that of computing the average ED Time to Physician Initial Assessment (PIA) in a hospital. This permits understanding how long patients wait before being seen by a physician (and thus before any diagnostic procedures or interventions are being prescribed). To provide meaningful query responses, we need a complete business schema instance. An initial course of action may be that of vivifying incomplete or uncertain PIA times by using a default value. Such default value may vary by type of hospital (e.g., community vs. teaching), patient's acuity level and ED volume, and may be input into our process as a vivification assumption. This vivification strategy may be sufficient to interpret the operational data at hand within a local business environment, but it might be deemed unsuitable if later on, the query response is attached to a pay-for-performance funding model.

Under this new scenario, the government may require stricter strategies for imputing missing values, or may deem imputation impermissible. By recording vivification decisions, our framework is able to accommodate real-time changes and revise any decisions made during the interpretation process. This may involve adopting a different vivification strategy that counts in the indicator only patients for whom the database contains complete information.

Example 3. As another example, consider how vivification offers insight into how to tally the ED patient counts in a hospital. Initially, we may attempt to identify patients based on the concept of ED Visit, as suggested by DART guidelines. We may choose to vivify patient identifiers by generating unique identifiers (i.e., Skolem terms) from existing ED Visit identifiers. However, this strategy might introduce a bias and business users might need to understand from the vivified mapping how patients are counted. Nowadays, business users are mostly given numbers but not the intuition or rationale about the formulae used. A frequently ignored fact, which influences patient counts, is that of having patients visiting the ED more than once in a single day. An improved understanding of this situation may then cause a revision of the entity resolution strategy used on patients, and the adoption of one that successfully separates patients from their multiple ED visits. If we add the date of a hospital visit to patient identifiers to indicate a visit, we can distinguish multiple visits, but only on different days. The business user needs to decide if this is sufficient or if multiple visits on a single day need to be counted separately. This type of decision should not be hardcoded in procedural logic decided by the IT department. Our framework actively supports the declarative representation and modification of such decisions by a business user.

6 Related Work

Our work builds upon the foundation of knowledge base vivification [13] and data exchange between disparate databases [11,17]. The data-to-business interpretation semantics that we propose in this paper adapts the notion of certain answers to a vivification context [3]. Close in spirit to our work is that of Sismanis et al. [19]. This work proposes a resolution-aware query answering model for OLAP applications, where uncertainty about business entities is handled dynamically at query time; unlike ours, the work does not discuss how to deal with incomplete or uncertain attribute values in a general BI context. The problems of duplicate detection (a.k.a entity identification, data deduplication or record linkage) and data fusion have prominent roles in the data integration literature [8,9]. Numerous techniques have been proposed for vivifying real-world entities (deduplication) and their multiple possibly inconsistent descriptions (data fusion). Most of these cleansing techniques are still applicable in our proposed framework notwithstanding, the main differences between the two appear to be first, the point of intervention at which the techniques are being applied and second, our emphasis on declaratively recording these resolution decisions. In particular, we view mapping rules as crucial tools for recording and reasoning about incompleteness and uncertainty. Among recently studied information quality criteria,

Naumann's completeness of data measures the amount of `non-NULL` values in a database instance (this is done in comparison to a potentially complete universal relation) [16]. This criterion embodies a model that can also be used to calculate the completeness of query answers which result from combining sources through merge operators; as such, this work can be leveraged in our framework to encode specific completeness assumptions or constraints over business interpretations. Our work also relates to both data exchange and semantic web efforts on bridging open and closed world reasoning [14,18]. Moreover, our work complements that of van der Aalst et al. [20] in Business Process Modeling (BPM). While their emphasis is on discovering and enhancing business process models from event logs in the data, ours is on providing an interpretation from observed events with respect to business models. Last, to the best of our knowledge, current BI/ETL tools offer little to no support for systematically using *adaptive* incompleteness rules within their data analysis and interpretation processes. Indeed, a primary motivation for our work is the desire to put the many stages of BI processing (e.g., data acquisition, consolidation, analysis, and evaluation of queries) into a framework that openly and declaratively records at every BI stage, the assumptions and choices made to deal with incompleteness and uncertainty. The interpretation of `NULL` values must be done in the context of the answers business users seek, i.e., it cannot be done solely in a batch offline mode during the consolidation stage.

7 Concluding Remarks

In bridging the gap between the world of IT-supplied data and the world of business, we are working on a number of techniques for transforming operational data into complete high-quality business information. Current BI solutions tend to adopt a two phase approach to solve this problem, where the mapping of data generally precedes the process of imparting a business semantics to it. In this paper, we have advocated on-demand interpretation of IT-supplied data in terms of business concepts.

As for our future work, in one direction, we are developing a general mapping framework for embracing incompleteness. We are investigating how to determine, given an incomplete business mapping and a set of business queries, when is it possible to create a vivid business schema instance. We study a number of business query types and how rewritten mappings can be used to satisfy these queries. We also propose a declarative mechanism for expressing preference rules that aims at encapsulating default domain assumptions. In another direction, we plan to finalize our case study that implements and evaluates our vivification techniques.

Acknowledgements. The authors wish to acknowledge Alex Borgida for his insightful comments and suggestions, and also the anonymous reviewers for their feedback. This work was partially supported by the NSERC Business Intelligence Network.

References

1. Understanding Emergency Wait Times: Who is Using Emergency Departments and How Long are They Waiting. Canadian Institute for Health Information (2005)
2. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
3. Abiteboul, S., Kanellakis, P.C., Grahne, G.: On the Representation and Querying of Sets of Possible Worlds. Theor. Comput. Sci. 78(1), 158–187 (1991)
4. An, Y., Borgida, A., Miller, R.J., Mylopoulos, J.: A Semantic Approach to Discovering Schema Mapping Expressions. In: ICDE, pp. 206–215 (2007)
5. Arocena, P.C., Miller, R.J., Mylopoulos, J.: Vivification in Business Intelligence. In: Ng, R.T. (ed.) Perspectives on Business Intelligence. Synthesis Lectures on Data Management, vol. 5(1), pp. 33–52. Morgan & Claypool (2013)
6. Azvine, B., Cui, Z., Nauck, D., Majeed, B.A.: Real-Time Business Intelligence for the Adaptive Enterprise. In: IEEE CEC/EEE (2006)
7. Barone, D., Topaloglou, T., Mylopoulos, J.: Business Intelligence Modeling in Action: A Hospital Case Study. In: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 502–517. Springer, Heidelberg (2012)
8. Bleiholder, J., Naumann, F.: Data Fusion. ACM Comput. Surveys 41(1) (2008)
9. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate Record Detection: A Survey. IEEE Trans. Knowl. Data Eng. 19(1), 1–16 (2007)
10. Fagin, R., Haas, L.M., Hernández, M., Miller, R.J., Popa, L., Velegrakis, Y.: Clio: Schema Mapping Creation and Data Exchange. In: Borgida, A.T., Chaudhri, V.K., Giorgini, P., Yu, E.S. (eds.) Conceptual Modeling: Foundations and Applications. LNCS, vol. 5600, pp. 198–236. Springer, Heidelberg (2009)
11. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data Exchange: Semantics and Query Answering. Theor. Computer Science 336(1), 89–124 (2005)
12. Jiang, L., Barone, D., Amyot, D., Mylopoulos, J.: Strategic Models for Business Intelligence. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 429–439. Springer, Heidelberg (2011)
13. Levesque, H.J.: Making Believers out of Computers. Artif. Intell. 30(1), 81–108 (1986)
14. Libkin, L.: Data exchange and Incomplete Information. In: PODS, pp. 60–69 (2006)
15. Marnette, B., Mecca, G., Papotti, P., Raunich, S., Santoro, D.: ++Spicy: an OpenSource Tool for Second-Generation Schema Mapping and Data Exchange. PVLDB 4(12), 1438–1441 (2011)
16. Naumann, F.: Quality-Driven Query Answering. LNCS, vol. 2261. Springer, Heidelberg (2002)
17. Popa, L., Velegrakis, Y., Miller, R.J., Hernández, M.A., Fagin, R.: Translating Web Data. In: VLDB, pp. 598–609 (2002)
18. Sengupta, K., Krisnadhi, A.A., Hitzler, P.: Local Closed World Semantics: Grounded Circumscription for OWL. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N., Blomqvist, E. (eds.) ISWC 2011, Part I. LNCS, vol. 7031, pp. 617–632. Springer, Heidelberg (2011)
19. Sismanis, Y., Wang, L., Fuxman, A., Haas, P.J., Reinwald, B.: Resolution-Aware Query Answering for Business Intelligence. In: ICDE, pp. 976–987 (2009)
20. van der Aalst, W.M.P.: Process Mining - Discovery, Conformance and Enhancement of Business Processes. Springer (2011)

An On-Demand ELT Architecture for Real-Time BI

Tobias Freudenreich[1], Pedro Furtado[2], Christian Koncilia[3], Maik Thiele[4], Florian Waas[5], and Robert Wrembel[6]

[1] Technische Universität Darmstadt, Germany
freudenreich@dvs.tu-darmstadt.de
[2] University of Coimbra, Portugal
pedro@u-coimbra.pt
[3] University of Klagenfurt, Austria
koncilia@isys.uni-klu.ac.at
[4] Technische Universität Dresden, Germany
maik.thiele@tu-dresden.de
[5] Greenplum/EMC, San Mateo, U.S.A.
flw@greenplum.com
[6] Poznań University of Technology, Poland
Robert.Wrembel@cs.put.poznan.pl

Abstract. Online or real-time BI has remained elusive despite significant efforts by academic and industrial research. Some of the most prominent problems in accomplishing faster turnaround are related to the data ingest. The process of extracting data from source systems, transforming, and loading (ETL) it is often bottlenecked by architectural choices and fragmentation of the processing chain.

In this paper, we present a vision for a resource-efficient infrastructure that enables just-in-time processing with regards to data ingest. At the heart of our approach are (1) the converting of compute intensive parts of the ETL process into in-database processing and (2) the activating of the process on demand via a system of flexible views.

Our approach avoids processing of data that is not being accessed any time soon, scales effectively with the database system and avoids administration and management overhead.

Keywords: Real-time BI, ETL, ELT, Materialized Views.

1 Introduction

While scalability and performance enhancements for data warehouse technology have advanced greatly over the past decade, BI architectures are increasingly bottlenecked on their data ingest. The prevailing paradigm of managing data ingest and analytics with disparate infrastructures—the former with dedicated ETL systems, the latter with a centralized enterprise data warehouse—means that data ingest to the data warehouse poses a substantial obstacle that needs to be overcome in order to provide real-time BI services. The breakdown into two

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 50–59, 2013.

separate phases, an ETL phase and a querying or reporting phase, means that business insights are always delayed by the elapsed time required to complete the entire ETL processing first.

This problem is of high practical relevance when it comes to delivering critial business insights in short order: over the past decade certain industry verticals have come to rely on the rapid analysis of sales data and the application of predictive analytics to them to help make instant business decisions. Being able to deliver this information in on-line or real-time fashion promises to be an enormous competitive advantage.

In order to overcome the ingest bottleneck the ETL process needs to be broken up so that some data can be made available to querying immediately while the processing of other data continuous, *i.e.*, is not yet complete. However, this requires the ETL process to be made cognizant of the down-stream requirements at any time and process data with appropriate priorities. Current ETL architectures and infrastructures are inhibiting such processing and the currently prevailing practice of provisioning, maintaining and administrating entirely disparate systems for ETL and data warehouse technology poses significant obstacles to the desired just-in-time processing.

In previous work, efforts focused mainly on speeding up ETL as a separate process and significant progress has been made in parallelization and clustering of ETL systems so far. However, the resulting infrastructure, although of high throughput, remains strictly decoupled from its consumers and cannot react to specific demands. By partitioning the incoming data in smaller and smaller batches, the overall elapsed time of each individual ETL phase can be shortened and priorities may be assigned to certain batches statically. However, priorities implemented as static assignments can only alleviate the situation if the access pattern is known *a priori*. Ad-hoc queries or changing business objectives cannot be taken into account.

In this paper, we describe a vision for just-in-time ETL that provides data ingest with automatic prioritization of the data that is required by the down-stream logic of queries over the data warehouse. The key ingredients of our solution are:

1. Turn ETL into ELT, *i.e.*, load data directly into the data warehouse without external transformations, data cleansing or normalization;
2. Express the transformation logic conventionally implemented by the ETL platform as SQL in the form of views that can be directly referenced by OLAP queries;
3. Intelligently materialize the views and maintain incrementally as new data arrives so as to minimize refresh times;

The resulting architecture is distinguished by a minimal resource footprint and fully integrated in the data warehouse infrastructure. Any performance enhancements to the data warehouse apply directly to the ELT process. Moreover, the target architecture does not require separate provisioning, administration or maintenance overhead. The approach is constrained by the expressiveness of

materialized views—which differs from system to system—and as part of this work, we list several open problems that need to be addressed to make this approach mainstream.

Roadmap. The remainder of this paper is organized as follows: In Section 2 we briefly review the technical background and conventional best practices regarding ETL and data warehousing. In Section 3 we illustrate how views and their materialization form a lean, on-demand ELT infrastructure. In Section 4 we provide pointers to open research problems and conclude the paper with Section 5.

2 Background

Before discussing our approach we briefly review the basic concepts of conventional ETL-based data warehousing architectures and discuss their limitations. For a in-depth discussion we refer the interested reader to [9], [2], [4], [5].

2.1 Conventional ETL-Based Architecture

Figure 1 provides an overview of the different components and classes of infrastructure that make up the overall architecture of a data warehousing implementation. We distinguish the following major groups:

Operational Data Stores. Any data management system involved in data acquisition or generation can be an *operational data store (ODS)*. Typically, transaction processing constitutes the majority of ODS's for sales data warehousing. However, other operational data such as click streams, web logs or web crawls may well be considered too. In practice, an enterprise's ODS component may comprise hundreds or even thousands of systems of rather heterogeneous nature both in terms of their implementation as well as the data they produce.

ETL Tools. In order to make the data centrally available for analysis, the ODS's output data needs to be consolidated first. During the ETL phase, data is extracted from the individual ODS's, transformed—*i.e.*, homogenized with respect to a desired schema, cleansed and potentially preprocessed in business-specific ways—and subsequently loaded into the data warehouse. While extraction and loading are usually a matter of standardized or proprietary connectivity tools such as ODBC, JDBC, etc., the transformation of data is more of a free-form process and has to take into account business rules and conventions. The transformation of data during ETL is often viewed as a workflow processing problem. As a result, a whole ecosystem of tools has developed in this space. The more advanced tools offer sophisticated GUI-based methods to define and visualize processing steps, however, in almost all enterprises pockets of hand-coded scripts for the transformation of operational data are present too. Historically, ETL tools bridged the gap between ODS's and the data warehouse also with respect to hardware resources: a dedicated ETL infrastructure buffered input and output data and made for a much needed decoupling of often resource restricted data

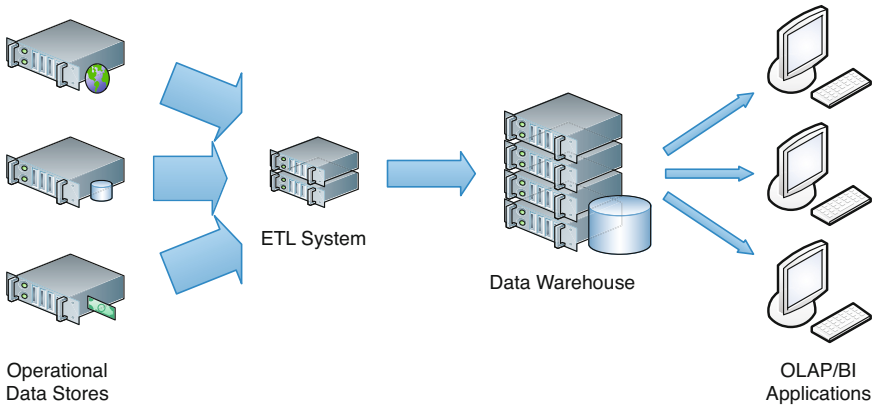


Fig. 1. Conventional ETL-based architecture. A separate ETL component consitutes a scalability and throughput bottleneck in addition to operational overhead. Downstream workflows are constrained by the data ingest limitations.

warehouse infrastructure and the fire hose-like ODS's. As a result, stand-alone ETL systems emerged that were provisioned for and maintained separately.

Enterprise/Analytical Data Warehouse. Lastly, the target platform of the entire data processing as discussed so far is making the data available for reporting and/or analysis through a data warehouse. Traditionally more used for reporting, a new breed of data warehouse technology has emerged over recent years: the analytical data warehouse. Similar in concept to the classic enterprise data warehouse, analytical data warehouses are meant to enable data scientists to perform much deeper and more compute-intensive analysis. These systems are high-powered scalable database systems, often in the form of MPP clusters (see *e.g.* [7]).

2.2 Critique of Conventional Approach

The architecture presented above is well-established technology and constitutes a multi billion-dollar business today. In the following we will scrutinize different aspects of ETL and highlight their pros and cons.

Delayed Data Availability. Historically, data processing cycles became aligned with the business day: all data of one business day is processed over night and loaded for reporting and analysis on the next day. This type of processing is highly desirable from an operational point of view as such a rhythm simplifies the processing and makes for very tangible deadlines. With its strict boundaries of ETL logic delineated from data sources upstream and data warehouses downstream, ETL tools have been developed at their own pace and largely independent of any of the producer or consumer systems. Consequently, the business cycle and with it the delay until the next day has become more and more cemented and batch processing is the *de facto* standard in data warehousing to

date. Enterprises with higher requirements for data freshness have resorted to processing of so called *micro batches*. This is still the same type of processing just revved at a higher pace, *i.e.*, small batches are processed by the ETL infrastructure at higher frequency.

Unnecessary Processing of Unused Data. Since ETL processing is driven by the availability of data upstream but does not take into account what data is required downstream, the tools end up processing *all* available data. While simple and practical, this approach wastes potentially significant resources on processing data that is not needed most urgently, or may not be needed ever. As a result, resources are not used optimally and again precious time may be wasted waiting for processing data that is not required.

Limited Scalability. A decoupled ETL infrastructure presents the risk of bottlenecking the entire data ingest process. The data volumes to process per cycle are not constant over time. For example, data volumes for the 4th quarter at a retailer outweigh other quarter significantly. And even within a quarter, certain periods such as the Monday after Thanksgiving may produce significantly more operational data than other days within the same week. To allow for such fluctuations, a classic architecture must provision spare capacities that will sit idle most of the time but guarantee survival during such peak times.

Limited Expressiveness. Over the past decade ETL platforms have more and more developed into workflow processors with substantial expressiveness in the language they employ. Yet, not having unfettered and efficient access to the data downstream makes certain transformations and cleansing operations difficult or impossible to implement.

Operational Overhead. Having to operate and maintain one or more ETL systems in addition to a data warehouse infrastructure puts additional strain on the enterprise's IT organization. Maintenance operations need to be synchronized carefully with the consumers resulting in multiplied efforts across the architecture and significant operational overhead.

In summary, established ETL architectures may serve a number of standard scenarios very well but as we see business requirements call for more flexible and timelier data ingest to provide predictive analysis and up-to-the-minute reporting, these architectures become increasingly performance bottlenecks and an operational burden.

2.3 Other Architectural Choices

Besides classic ETL processing an number of other approaches have been suggested in light of real-time processing requirements. In particular, *Data Stream Management Systems (DSMS)* and *Complex Event Processors (CEP)* are technologies developed specifically for rapid processing and high data throughput [3]. They constitute a trade-off between expressiveness and timeliness that can be summaries as follows: the more timely the result the less actual processing is possible (see Fig. 2). Stream processing systems and CEP engines are highly

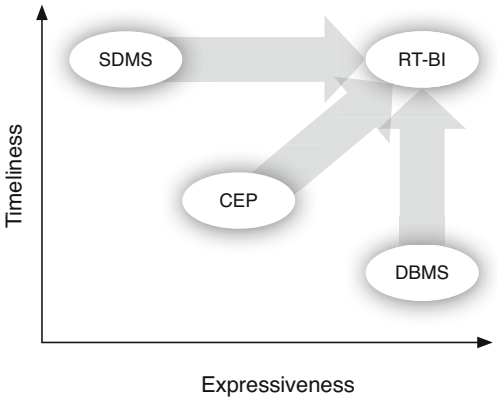


Fig. 2. Categorization of data management systems with regards to expressiveness and timeliness

suitable for application scenarios that require alerting and very limited data analysis. However, the currently available commercial systems appear unsuitable for real-time data ingest due to their severely limited processing capabilities. A true real-time ETL system will have to provide characteristics similar to a DBMS yet with the timeliness of CEP or DSMS systems as shown in the upper right corner in Figure 2.

3 On-Demand Processing for Real-Time BI

Based on the critique of the established processing conventions, we have designed an architecture that overcomes the drawbacks of classic ETL processing as listed above. Our approach consists of two fundamental principles, (1) convert ETL into ELT processing and consolidate the underlying systems and (2) materialize the results only on-demand instead of up-front.

3.1 Converting ETL into ELT

Conceptually, transformations executed during ETL are data modifications that can be expressed in a number of suitable languages. SQL was primarily designed for querying data and hence has the full repertory for data access as well as data manipulation—the latest SQL standards achieved Turing completeness, *i.e.*, there is no restriction regarding the expressibility of business logic.

Converting ETL into ELT means loading the data first into "scratch space" within the data warehouse and executing transformation steps via SQL afterwards. This approach has been suggested in the past and received a substantial amount of attention [2]. Past approaches focused on the expressiveness and the general feasibility. We consider these aspects solved problems by now regardless of the level of sophistication of the original ETL process. When simply converting ETL to ELT transformations encoded as SQL scripts and procedures would

be activated by a scheduler to process batches of data similar to external ETL processing.

However, ELT processing instead of ETL has not caught on across the industry in the past for a couple of reasons. First, many conventional data warehouses are simply transactional database systems operated as analytic data warehouses. That is, they are not designed to handle significant workloads nor scale very well. Adding an additional ELT processing workflow to the already strained database may cause performance degradation for various downstream users. Similarly, transaction processing databases provide only limited scalability in terms of data volumes. Again, adding ELT processing to an already constrained system results in operational difficulties. Specialized data warehousing systems overcome these obstacles and provide high-powered query capabilities as well as scalability that enables the conversion of ETL to ELT relatively easily. We anticipate seeing tools vendors embrace this trend and provide elaborate GUI-based development environments—similar to those currently available for stand-alone systems—in the future.

The sheer conversion of ETL to ELT usually results in a strong boost for the timeliness of data processing already. Analytic data warehouse systems provide significantly higher processing power than stand-alone systems and can ingest and process data at substantially higher rates. As mentioned above this could be viewed as a previous approach supplemented with a more powerful platform resulting in better execution. It is a stepping-stone but does not provide real-time processing *per se*, just yet.

3.2 On-Demand Materialization of Processed Data

Our approach takes this method further. Instead of expressing transformations as SQL procedures that are activated by a scheduler, we express the transformation logic in the form of *views*. For modularity and practical purposes the views may be stacked, *i.e.*, views referencing other views. This way, transformations are performed *on-demand* whenever the views are referenced in analytic or reporting queries. In this scheme, the loading phase is no longer needed as the view definition makes the data available as needed. Using external data access mechanisms such as *External Tables* [11], [7], which stream data from external sources and make them available as regular tables, even the extraction part can be incorporated in the on-demand processing.

As a result, the processing of both extraction and transformation can leverage the processing capabilities of the data warehouse and scale to maximum throughput. The views limit the data accessed and therefore imported to the database to the minimally required amount which guarantees the smallest possible data footprint and conserves processing resources, further speeding up the data pipeline. A sophisticated query optimizer may further limit the processing through the stack of views.

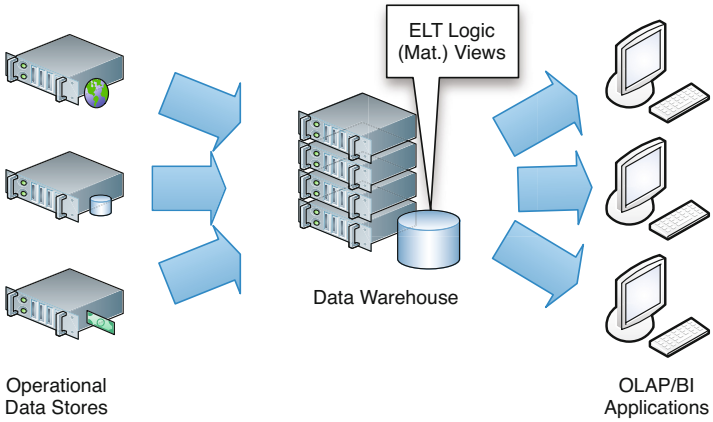


Fig. 3. On-demand ELT architecture with scalable ELT in the form of views and/or materialized views as part of the data warehouse

Lastly, we propose to materialize the views and incorporate view maintenance on-demand, that is, check and refresh views as needed whenever queries reference them. Highly efficient incremental view maintenance algorithms that do not require large recomputation but solely a recomputation based on the difference of the base tables have been suggested in the literature [8], [12], [6] although a number of implementation details remain open problems as we will see in the next section. A sketch of the resulting architecture is presented in Figure 3.

Currently, a prototype for the architecture as outlined above is underway using a commercial database system. We hope to be able to report on specific results very soon.

4 Open Research Questions

The design of our real-time BI architecture is primarily based on the notion of materialized views and the idea that the cost of the initial materialization as well as the refreshing or maintaining is amortized over the course of workload.

Materialized views are a mature technology in today's database products. The currently available mechanisms for keeping them up-to-date vary widely between products along two dimensions: (1) incremental maintenance or maintenance through recomputation, and (2) automatic or manual trigger of maintenance.

Recomputation is the simplest—and usually most costly—of all maintenance modes: the existing materialization is discarded and the view is recomputed from scratch. That is, recomputation is always an applicable maintenance option. Incremental maintenance refers to the notion that individual changes to the inputs of the view can be applied to the view and only those rows in the view that are affected are modified. This type of maintenance is only applicable to views that meet certain criteria. In practice, incrementally maintainable views

cover a significant portion of the space; so much so that some vendors *exclusively* offer incrementally maintainable views [10].

The biggest challenge our approach presented in the previous section poses is that of materializing and maintaining views on-the-go, *i.e.*, without explicit declaration and without explicit triggering of maintenance. In particular, this requires further investigation into:

View Selection. Identification of expressions that should be materialized must be interleaved with query activity. Current state-of-the-art of recommenders or database tuning tools requires workloads to be processed off-line. We envision the decision which views to materialize to be made—and frequently revisited—during regular querying.

View Maintainance. Conventional change detection mechanisms rely on logic embedded in the underlying data manipulation operations, *e.g.*, regular DML operations, bulk loading, or data-affecting DDL operations. Ingesting data from ODS's poses a number of new challenges and requires integration with external notification mechanisms or better techniques for detecting changes on external data sources.

Finally, an entirely different subject is the quantitative analysis of such an approach: currently there are no benchmarks that gauge the timeliness of such an architecture end-to-end. A number of performance benchmarks can be used to check for the effectiveness of the views, however, the upstream data ingest is not assessed with such methods. Initial results have been presented in [13] and could be developed into a more comprehensive framework for quantitative analysis.

We believe initial results in the areas outlined above are within reach—others will require more research. In either case, the overall architectural proposal seems to warrant to start investigating these aspects.

5 Summary

In this paper, we presented an outline for a real-time BI architecture based on two simple yet powerful principles: (1) instead of treating ETL as a separate processing phase executed on independent infrastructure we propose moving it into the actual data warehouse in the form of ELT expressed as views, *i.e.*, loading raw data and transforming it using the processing capabilities of the data warehouse, and (2) materializing and maintaining these views on demand.

The resulting approach promises significantly better resource utilization as only truly required data is being processed and avoids having to wait for a pre-processing phase to be completed before data can actually be accessed through reporting and analytical processing.

Our approach supposed flexibility in the way materialized views are created and maintained that is currently not available in commercial products but opens up various alleys for future research that we are eager to pursue.

Acknowledgments. The authors would like to thank the organizers and participants of the BIRTE workshop for their constructive feedback and discussions around this subject. The work of Robert Wrembel is supported from the Polish National Science Center (NCN), grant No. 2011/01/B/ST6/05169. Tobias Freudenreich's work is supported by the LOEWE Priority Program Dynamo PLV (`http://www.dynamo-plv.de`).

References

1. Agrawal, D., Abbadi, A., Singh, A., Yurek, T.: Efficient View Maintenance at Data Warehouses. SIGMOD Record 26(2), 417–427 (1997)
2. Andzic, J., Fiore, V., Sisto, L.: Extraction, Transformation, and Loading Processes. In: Wrembel, R., Koncilia, C. (eds.) Data Warehouses and OLAP: Concepts, Architectures and Solutions, pp. 88–110. IGI Global (2007)
3. Buchmann, A., Koldehofe, B.: Complex Event Processing. Information Technology 51(5), 241–242 (2009)
4. Bruckner, R.M., List, B., Schiefer, J.: Striving Towards Near Real-time Data Integration for Data Warehouses. In: Kambayashi, Y., Winiwarter, W., Arikawa, M. (eds.) DaWaK 2002. LNCS, vol. 2454, pp. 317–326. Springer, Heidelberg (2002)
5. Chaudhuri, S., Dayal, U., Narasayya, V.: An Overview of Business Intelligence Technology. Communications of the ACM 54(8), 88–98 (2011)
6. Colby, L.S., Griffin, T., Libkin, L., Mumick, I.S., Trickey, H.: Algorithms for Deferred View Maintenance. SIGMOD Record 25(2), 469–480 (1996)
7. Greenplum Inc.: Technical Documentation, `http://www.greenplum.com`
8. Gupta, A., Mumick, I.S.: Materialized Views: Techniques, Implementations, and Application, 1st edn. MIT Press (1999)
9. Kimball, R., Caserta, J.: The Data Warehouse ETL Toolkit. John Wiley & Sons Inc. (2004)
10. Microsoft Corporation. Microsoft SQL Server, Technical Documentation, `http://www.microsoft.com/sql`
11. Oracle Corporation: Technet. Technical Documentation, `http://www.oracle.com/technetwork/indexes/documentation/index.html`
12. Quass, D., Widom, J.: On-Line Warehouse View Maintenance. In: Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 393–404 (1997)
13. Vassiliadis, P., Karagiannis, A., Tziovara, V., Simitsis, A.: Towards a Benchmark for ETL Workflows. In: Proc. of Int. Workshop on Quality in Databases (QDB), pp. 49–60 (2007)

Instant-On Scientific Data Warehouses

Lazy ETL for Data-Intensive Research

Yağız Kargın, Holger Pirk, Milena Ivanova,
Stefan Manegold, and Martin Kersten

Centrum Wiskunde & Informatica (CWI), Amsterdam, The Netherlands

Abstract. In the dawn of the data intensive research era, scientific discovery deploys data analysis techniques similar to those that drive business intelligence. Similar to classical Extract, Transform and Load (ETL) processes, data is loaded entirely from external data sources (repositories) into a scientific data warehouse before it can be analyzed. This process is both, time and resource intensive and may not be entirely necessary if only a subset of the data is of interest to a particular user. To overcome this problem, we propose a novel technique to lower the costs for data loading: *Lazy ETL*. Data is extracted and loaded transparently on-the-fly only for the required data items. Extensive experiments demonstrate the significant reduction of the time from source data availability to query answer compared to state-of-the-art solutions. In addition to reducing the costs for bootstrapping a scientific data warehouse, our approach also reduces the costs for loading new incoming data.

1 Introduction

Thirty years of database research knowledge is slowly finding its way into the domain of science [7]. This trend is driven by the need to handle the large amounts of data that are the result of increasingly automated acquisition of scientific data. Especially life and earth sciences obtain more and more data ever faster and faster. The capacity to sequence genomes, e.g., has been outpacing Moore's Law in the last years and is expected to continue to do so [24].

In addition to the increasing data volume, the character of science itself changes. In the past, researchers used to form hypotheses and validate them using experiments. Today they first run cheap, high throughput experiments and mine the results for "interesting" knowledge. This became known as eScience and culminated in the formulation of *the fourth paradigm* [7]. In its essence, this process (see Figure 1) is similar to that which drives business intelligence.

It is, therefore, natural to consider the techniques that helped meet the requirements of business intelligence to help scientists in their daily work. A significant amount of research has been done on various aspects of eScience. This covers problems of querying [18], mining [13] and visualization [19]. However, all of these assume that the underlying data is readily available in the scientist's database. This is generally not the case.

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 60–75, 2013.

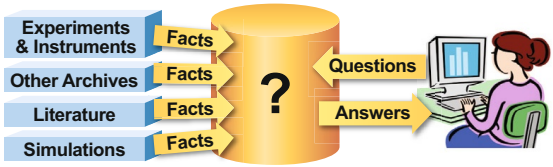


Fig. 1. EScience Knowledge Management (taken from [7])

Instead, input data is usually collected in domain-specific files and repositories (i.e., semi-structured collections of files). The need to physically load all data into a data warehouse system before analysis forms a burden that domain scientists are reluctant to accept. This burden is exacerbated by the fact that few scientists have access to data integration specialists. In general, scientists have to take care of their data management needs themselves. Thus, they abstain from "superior solutions" that require significant data management knowledge. Our objective is to provide eScientists with a data ingestion system that is easy to use, extensible and scalable to many terabytes of data. To this end, we present a query-driven, on-demand Extract, Transform & Load (ETL) system that minimizes the burden of data ingestion.

To achieve the necessary scalability, our approach limits the initial loading to the metadata of the input files. The actual data is extracted, transformed and loaded transparently at query time for the files containing the data that the queries require. In addition to reducing the cost for bootstrapping a scientific data warehouse for a new file repository, our approach also makes updating and extending a warehouse with modified and additional files more efficient.

We implemented our novel *Lazy ETL* system in MonetDB [1], an open-source column-store Database Management System (DBMS). To showcase our solution, we use typical analysis tasks on seismological sensor data that is available in mSEED files and a file repository as data source, which is one of the kinds of source datastores that ETL processes deal with [23]. Extensive experiments demonstrate the significant reduction of the overall time from source data availability to query answer compared to state-of-the-art solutions that require the ETL task to be completed for the entire data before running queries. Even though we focus on a particular scientific use case, we believe that other cases, like classical business ETL, can benefit from a similar approach, as also mentioned by Dayal et al. [5] and Haas et al. [6].

We organized the rest of this paper as follows: In Section 2 we describe the state of the art in data ingestion and discuss its shortcomings with particular focus on scientific data management. In Section 3 we present our architecture for overcoming these shortcomings without losing the merits of traditional ETL. In Section 4 we evaluate our solution in a scientific case, and we draw conclusions in Section 5.

2 State of the Art in Data Ingestion

As explained, the data ingestion needs of eScientists are very similar to those of business analysts. We, therefore, describe the state of the art of data ingestion in business data warehouses before we explore the transferability into the eScience domain. EScientists load data from sensor data files much like business warehouses load data from Online Transaction Processing (OLTP) systems.

The most popular setup to serve transactional and analytical applications is the following [4]: a row oriented operational DBMS for the OLTP-load and a Data Warehouse for the analytical needs. New data enters the operational system as it occurs and is loaded into the Warehouse in intervals using an *ETL* process [14]. This, however, has several drawbacks:

1. The data that has not been transferred to the OLAP-store yet, will not appear in the aggregated results, which renders the OLAP-store constantly out of date [17].
2. All data has to be held redundantly which increases the costs for hardware acquisition and maintenance [14].
3. The update process has to be maintained and run periodically to keep the OLAP-store reasonably up to date. Since this process can be complicated the added costs in hardware and personnel can be high [14].

The costs may increase even further with the complexity of the management's requirements. A common requirement that is especially interesting is real time reporting. Vendors support this through means of *Active Warehousing*.

2.1 Active Warehousing

To increase the efficiency of business operations it is often required to do analytics in a relatively short period of time (an hour or even minutes). This kind of *Operational Reporting* [10] is a trend that has been recognized by vendors [3]. They aim at supporting it by means of *Active Warehousing*: Shortening of the update interval. This reduces the deviance of the aggregates from the real, transactional data and therefore allows almost real time reporting. It does however chronically increase the load on both the transactional and the analytical database. The transactional database has to handle additional extracts which cannot, as is common in traditional warehousing, be scheduled in the downtime of transactional operations but have to be executed concurrently to the transactional load.

2.2 Lazy Aggregates

The update interval in Active Warehouses is shorter than in traditional warehouses but still a constant. The deviance between the real data is therefore undetermined because it may be changed arbitrarily by a transaction unless special restrictions are implemented. A possibility to limit this deviance is provided by a technique known as *Lazy Aggregates* [15]. The warehouse update is

not triggered after a given interval but when the deviance exceeds a predefined threshold. This assumes that it is significantly faster to calculate the deviance that is induced by a processed transaction than to run the update. Depending on the aggregation function calculating the deviance without calculating the value can be costly or even impossible (e.g., for holistic functions). In that case this approach fails to yield any benefit.

2.3 Scientific Data Ingestion

We are not the first to recognize the need for solutions that support scientific data ingestion. In particular, the problem of loading data from external files has received significant attention from the database community. The SQL/MED standard (Management of External Data) [21] offers an extension to the SQL standard that simplifies the integration of external files into a relational schema. It allows the SQL-server to control referential integrity, authorization, and recovery for data in external files. However, it still puts the burden of managing the files on the user. Using the Oracle Database File System (DBFS) [16], files with unstructured data are transparently accessible as BLOB-attributes of relational tuples. But DBFS provides no structured view to external file contents. Similarly, there are also well-established techniques like external tables in any major commercial system. These provide us access to data in external sources as if it were in a table in the data warehouse. However, external tables require every query to access the entire data as opposed to Lazy ETL accessing only the data that queries require. DBMS query processing over flat data files is proposed by Idreos et al. [8]. They provide query-driven on-demand loading as we do. However, they limit the format to those that map straightforward to relational tables, such as CSV and tables in Flexible Image Transport System (FITS). Consequently they cannot handle more complex file formats that are common in eScience applications like MiniSEED (mSEED) or Geo Tagged Image File Format (GeoTIFF). These files contain complex schemas of different types of objects and even pointers that should be resolved to foreign key constraints when loading. The need for symbiosis of databases and file systems for eScience is in the core of the Data Vault concept presented in [11]. Our work can be seen as a further development of the idea of just-in-time access to data of interest integrated with the query processing.

Hence, none of the above provides an adequate solution to the data ingestion problem in eScience. In the following we present our approach to the problem: *Lazy ETL*. While targeted at scientific use cases, we believe that our solution is generally applicable to ingestion problems that involve (repositories of) external files.

3 Lazy ETL

Traditional ETL relies on proactively filling the data warehouse with data. However, a user may only need a subset of the available database. In the extreme

case, he might only want a fast answer to one ad-hoc query. Nevertheless, he still has to wait for the lengthy *eager* loading of all the input data. To mitigate this problem, only the "interesting" part of the data could be loaded. However, without knowledge of the workload it is unclear which subset of data is interesting. Hence, traditional ETL processes load and pre-aggregate all data that can possibly be queried into the data warehouse.

Breaking with the traditional paradigm, we consider ETL as part of the query processing. This allows us to run ETL only when required and only for the data that is required by a query. We call this approach *Lazy ETL*. To implement Lazy ETL, we create a virtual warehouse that is filled with data on demand. While executing queries, required data that is not yet present in the warehouse is loaded. This does not only facilitate the initial loading, but also the refreshing of the data warehouse contents. After the initial loading, refreshments are handled inherently when the data warehouse is queried.

In the following we illustrate how to replace a traditional ETL process with our lazy ETL scheme without loosing the benefits of the traditional approach. We visit each step of the lazy ETL process to discuss the further details.

3.1 Lazy Extraction

Extraction is the process of making the input data available for subsequent transformation and loading. Traditionally all data items that may be needed during transformation are extracted from the input data sources. In the typical ETL case with OLTP systems as data sources, structured queries may be used to limit the extracted data. This is particularly useful for incremental updates after the initial loading [22]. For flat file repositories, however, such optimization opportunities are limited to whatever the respective file format library provides. This intensifies the challenge of incremental updates and provides additional opportunities for lazy extraction.

Metadata. Without any knowledge about the input data files, all available files have to be considered "relevant" for a given query. Fortunately, many scientific file formats define the notion of *metadata*. Metadata is data that provides some insight into the content of a file but is still cheap to acquire. In an eScience scenario this covers data such as the time, the sampling rate or the position of a sensor. The costs to acquire such metadata is usually many orders of magnitude lower than loading the actual data. Metadata may even be encoded in the filename which would allow extraction without even opening the file. Therefore, we consciously decided to load the metadata eagerly. This initial investment will be amortized with the first query and provides the user with some overview of what to expect in the actual data. Although the definition of metadata might differ with file format, use case and system performance, in most of the cases it is relatively straightforward to decide which data to load eagerly. If no metadata is available for a given data source it may prove beneficial to manually define it.

Actual Data. We call all data that is not metadata actual data. In the eScience domain actual data items usually describe individual data points and come with,

e.g., a timestamp and one or many measured values. In practice, we expect the majority of the data items to be actual data. Actual data is, therefore, subject to lazy ETL. Although this query-driven on-demand extraction might cause overhead in source datastores, it is still only for a smaller set of data required for a query on the contrary to the eager incremental updates.

To select the files to load, the selection predicates on the metadata are applied. Once this part of the plan is executed, a rewriting operator is executed. This operator uses plan introspection and modification that are provided by the MonetDB system to replace all references to relational tables with operators that load the necessary files. This allows us to seamlessly integrate lazy extraction with query evaluation.

3.2 Lazy Transformation

An ETL process might contain transformations for different tasks, such as pivoting, one-to-many mappings, schema-level transformations etc. [22] [23]. Most of these transformations (e.g., projections, joins, etc.) can be provided by a relational query processor. It is therefore common practice to express these transformations in SQL. Transformations that use richer data mining can be potentially handled by user defined functions. This makes our approach similar to industrial common practice ELT (Extract, Load & Transform), but ELT does not have the laziness concept.

In our approach we implement all necessary transformations as relational views on the extracted data. This happens transparent to the user and has a number of benefits:

- It naturally supports lazy processing of the transformations because view definitions are simply expanded into the query. Only the minimal amount of needed data items from the sources are transformed.
- It allows the transformations to benefit from query optimization. These may even be based on information that is not known at "eager" load time (e.g., data distribution or selectivities).
- It provides an inherent means of data provenance since the transformations are still visible at query time.

3.3 Lazy Loading

Loading in ETL is the storing of the extracted and transformed data into the internal data structures of the data warehouse. This is largely done for performance reasons and is, thus, treated as an optimization in our system. From a query processing point of view, materialization of the transformed data is simply caching the result of the view definition. Our target system, MonetDB, already supports such caching in the form of *intermediate result recycling* [12].

Whilst not part of our current system, we consider integration of lazy ETL with intermediate result recycling a promising optimization and part of future work. This will make lazy ETL even lazier. Note that caching comes with the challenge of keeping the cache up to date.

4 Evaluation

To evaluate our approach, we have chosen seismology as the scientific domain for our use case. Our choice is mainly driven by a tight cooperation with project partners, seismologists from the Royal Dutch Meteorological Institute (KMNI). We strongly believe that other scientific domains have similar use cases, and that our techniques and result can thus be generalized to other science fields.

4.1 Data Source

Seismology is a scientific domain where huge amounts of data is generated with ever lasting movements of the Earth's surface. In seismology, the Standard for the Exchange of Earthquake Data (SEED)[2] is the most widely used standard file format to exchange waveform data among seismograph networks (e.g., transferring data from a station processor to a data collection center). A SEED volume has several ASCII control headers and highly compressed data records, i.e., the waveform time series. The control headers keep the metadata, which consists of identification and configuration information about the data records (i.e., actual data). In our experiments we use mSEED files, which contain less metadata. mSEED files are kept in large remote file repositories with direct FTP access [20].

4.2 Data Warehouse Schema

The normalized data warehouse schema for our use case is derived straightforwardly from the mSEED file format. An mSEED *file* contains multiple mSEED *records* (about 35 records per mSEED file on average in our data collection). An mSEED record contains the sensor readings over a consecutive time interval, i.e., a timeseries of about 3500 values on average in our data collection. Consequently, the normalized data warehouse schema — as given in Listing 1.1 — consists of three tables and one view. Tables `files` and `catalog` hold the metadata per mSEED file and mSEED record, respectively, while table `data` stores the actual sensor data. Each mSEED file is identified by its URI (`file_location`), and contains the metadata describing the sensor that collected the data (`network`, `station`, `location`, `channel`) as well as some technical data characteristics (`dataquality`, `encoding`, `byte_order`). Each record is identified by its (record) sequence number (unique per file), and holds metadata such as `start_time`, sampling rate (`frequency`), and number of data samples (`sample_count`). The `data` table stores the timeseries data as (`timestamp`, `value`) pairs. For user convenience, we define a (non-materialized) view `dataview` that joins all three tables into a (de-normalized) "universal table".

Listing 1.1. Data Warehouse Schema

```
CREATE SCHEMA mseed;

CREATE TABLE mseed.files (
  file_location STRING,          dataquality   CHAR(1),
  network       VARCHAR(10),     station       VARCHAR(10),
  location      VARCHAR(10),     channel       VARCHAR(10),
  encoding      TINYINT,         byte_order    BOOLEAN,
  CONSTRAINT files_pkey_file_loc PRIMARY KEY (file_location)
);

CREATE TABLE mseed.catalog (
  file_location STRING,          seq_no        INTEGER,
  record_length INTEGER,         start_time    TIMESTAMP,
  frequency     DOUBLE,          sample_count  BIGINT,
  sample_type   CHAR(1),
  CONSTRAINT catalog_file_loc_seq_no_pkey PRIMARY KEY (file_location, seq_no),
  CONSTRAINT cat_fkey_files_file_loc
    FOREIGN KEY (file_location)
      REFERENCES mseed.files(file_location)
);

CREATE TABLE mseed.data (
  file_location STRING,          seq_no        INTEGER,
  sample_time   TIMESTAMP,       sample_value  INTEGER,
  CONSTRAINT data_fkey_files_file_loc
    FOREIGN KEY (file_location)
      REFERENCES mseed.files(file_location),
  CONSTRAINT data_fkey_catalog_file_loc_seq_no
    FOREIGN KEY (file_location, seq_no)
      REFERENCES mseed.catalog(file_location, seq_no)
);

CREATE VIEW mseed.dataview AS
SELECT
  f.file_location, dataquality, network, station, location, channel, encoding,
  byte_order, c.seq_no, record_length, start_time, frequency, sample_count,
  sample_type, sample_time, sample_value
FROM mseed.files AS f
  JOIN mseed.catalog AS c
    ON f.file_location = c.file_location
  JOIN mseed.data AS d
    ON c.file_location = d.file_location AND c.seq_no = d.seq_no;
```

4.3 Sample Workload

Our sample workload consists of the eight queries presented in Listing 1.2 that reflect characteristic data analysis tasks as regularly performed by seismologists while hunting for "interesting seismic events". Such tasks range from finding extreme values over *Short Term Averaging* (*STA*, typically over an interval of 2 seconds) and *Long Term Averaging* (*LTA*, typically over an interval of 15 seconds) to retrieving the data of an entire record for visualization and visual analysis. Query 1 finds the maximum values (amplitudes) for a given channel (`BHN`) per station in the Netherlands (`NL`) during the first week of June 2010. Queries 2 and 3 compute the short term average over the data generated at Kandilli Observatory in Istanbul (`ISK`) via a specific channel (`BHE`). The difference between Query 2 and 3 is that Query 3 has an additional range predicate on the `start_time` of the records that is semantically redundant, but might help

Listing 1.2. Sample Queries

```
-- Query 1
SELECT station, MAX(sample_value)
FROM mseed.dataview
WHERE network = 'NL'
  AND channel = 'BHN'
  AND start_time > '2010-06-01T00:00:00.000'
  AND start_time < '2010-06-07T23:59:59.999'
GROUP BY station;

-- Query 2
SELECT AVG(sample_value)
FROM mseed.dataview
WHERE station = 'ISK'
  AND channel = 'BHE'
  AND sample_time > '2010-01-12T22:15:00.000'
  AND sample_time < '2010-01-12T22:15:02.000';

-- Query 3
SELECT AVG(sample_value)
FROM mseed.dataview
WHERE station = 'ISK'
  AND channel = 'BHE'
  AND start_time > '2010-01-12T00:00:00.000'
  AND start_time < '2010-01-12T23:59:59.999'
  AND sample_time > '2010-01-12T22:15:00.000'
  AND sample_time < '2010-01-12T22:15:02.000';

-- Query 4
SELECT channel, AVG(sample_value)
FROM mseed.dataview
WHERE station = 'ISK'
  AND sample_time > '2010-01-12T22:15:00.000'
  AND sample_time < '2010-01-12T22:15:15.000'
GROUP BY channel;

-- Query 5
SELECT channel, sample_time, sample_value
FROM mseed.dataview
WHERE station = 'ISK'
  AND sample_time > '2010-01-12T22:15:00.000'
  AND sample_time < '2010-01-12T22:18:00.000';

-- Query 6
SELECT station, MIN(sample_value), MAX(sample_value)
FROM mseed.dataview
WHERE network = 'NL'
  AND channel = 'BHZ'
GROUP BY station;

-- Query 7
SELECT sample_time, sample_value
FROM mseed.dataview
WHERE seq_no = 1
  AND file_location =
      '/.../knmi/ORFEUS/2010/152/NL_HGN_02_LHZ.2010.152.16.47.34.mseed';

-- Query 8
SELECT sample_time, sample_value
FROM mseed.dataview
WHERE seq_no = 1
  AND file_location =
      '/.../knmi/ORFEUS/2010/158/NL_OPLO_04_BHN.2010.158.09.26.51.mseed';
```

the query optimizer to pre-filter on the (small) `catalog` table before evaluating the predicate on `sample_time` on the (huge) `data` table. Query 4 computes the long term average over the data from the same station as Queries 2 and 3, but now over all available channels. Query 5 retrieves a piece of waveform at a given location, e.g., to visualize the data around a seismic event detected via Queries 2, 3, 4. Query 6 is similar to Query 1, but calculates both minimum and maximum for a given channel (`BHN`) per station in the Netherlands (`NL`) without restricting the time period. Queries 7 and 8 retrieve the time series data of an entire record from a given file, e.g., for visualization, or further analysis by an external program. The difference between Query 7 and 8 is that Query 8 asks for a file that does not exist, i.e., yields an empty result.

4.4 Experimental Setup

Our experimentation platform consists of a desktop computer equipped with a 3.4 GHz quad-core Intel Core i7-2600 CPU (hyper-threading enabled), 8 MB on-die L3 cache, 16 GB RAM, and a 1 TB 7200 rpm hard disk. The machine runs a 64-bit Fedora 16 operating system (Linux kernel 3.3.2).

A copy of the ORPHEUS mSEED file repository is stored on a local server and accessible via NFS. The extraction of (meta)data from mSEED files is realized with the libmseed library [9].

We use MonetDB [1] as data warehouse, and extended it with our Lazy ETL techniques.

Table 1. Datasets and their sizes

records per table			size				
`files`	`catalog`	`data`	mSEED	CSV	MonetDB	+keys	Lazy
5,000	175,765	660,259,608	1.3 GB	74 GB	13 GB	9 GB	10 MB
10,000	359,735	1,323,307,090	2.7 GB	148 GB	26 GB	18 GB	20 MB
20,000	715,738	2,629,496,058	5.5 GB	293 GB	50 GB	38 GB	36 MB

We create 3 different datasets of increasing size by randomly selecting, respectively, 5000, 10000, and 20000 files of the 161329 files from year 2010. Table 1 lists some characteristics of the 3 datasets used. (The whole ORPHEUS repository holds more than 3.5 million files collected since 1988.)

We compare the following 3 different ETL approaches.

EagerExt refers to the traditional variant where an external stand-alone program reads the mSEED files and extracts the data. The transformation step involves solving a few schema- and value-level problems, since an mSEED file has somewhat different representations of the same data (e.g., not normalized). Then the program writes 3 CSV files, one for each table of our data warehouse schema. These files are then bulk-loaded into the data warehouse (using the `COPY INTO` SQL statement of MonetDB).

EagerSvr moves the functionality to read mSEED files into the DBMS server. We extended MonetDB with the required functionality to read mSEED files and extract their data into the tables of our data warehouse schema. Like EagerExt, EagerSvr still *eagerly* reads and loads the data from all given mSEED files before querying can begin. However, it does not require to serialize the data into a CSV file and parse that again. Rather, the data from the mSEED files is directly loaded into the database tables inside the DBMS server.

Lazy refers to our new approach as presented in Section 3. Initially, the database server only extracts the metadata from all given mSEED files and loads it into tables `files` and `catalog`. Only during query evaluation the actual time series data is extracted and loaded at the granularity of an individual file into a temporary `data` table and required records are taken. For now, to keep the storage footprint of the database server at a minimum, the loaded data is discarded as soon and the query has been evaluated. While caching loaded data might avoid repeated loading of the same files, the chosen approach inherently ensures up-to-date data even if the original files are updated. A detailed study when and how to cache and update loaded data goes beyond the scope of this paper and is left for future work.

In order to analyze the costs and benefits of creating and using primary and foreign key constraints, we consider two variations of both EagerExt and EagerSvr. The first variant — *EagerExt-* / *EagerSvr-* — omits the primary and foreign key constraints, while the second variant — *EagerExt+* / *EagerSvr+* — creates the respective indexes after data loading, but before querying starts. The rationale is that creating primary and foreign key indexes, in particular for the foreign keys from `data` into both `catalog` and `files`, increases the data loading costs and storage requirements, but can speed up query processing. For Lazy, we do not build any indexes.

With the rather small `files` and `catalog` tables, primary and foreign key indexes do not show any significant performance advantage during query evaluation. For the `data` table, the foreign key indexes could only be built during query evaluation after the data is loaded on-the-fly. However, since creating a foreign key index basically means calculating the foreign key join, this does not yield any benefit.

4.5 Loading

Table 1 lists the characteristics of our three datasets as well as the size of the original mSEED files, the CSV files generated by EagerExt, the size after loading into MonetDB without primary and foreign key indexes, the additional storage required for the primary and foreign key indexes, and the size of the loaded metadata only in the Lazy case. Due to decompression, serialization into a textual representation and explicit materialization of timestamps, the CSV files are much larger than the mSEED files.

An interesting point is that our lazy ETL approach is more space-efficient than eager approaches. The total amount of data stored in the data sources and the data warehouse is significantly less for the lazy case than for the eager cases.

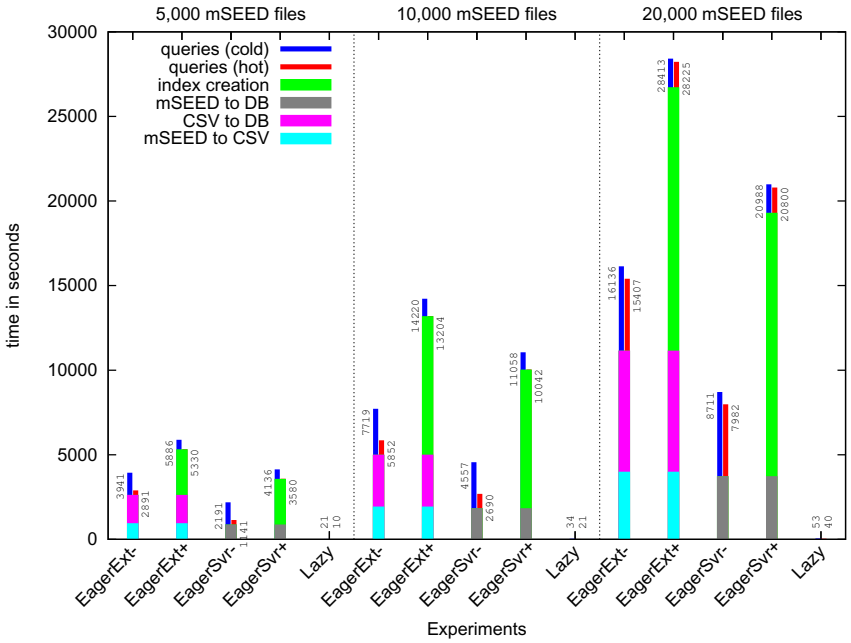


Fig. 2. Loading

Figure 2 breaks down the data ingestion costs into (a) extracting data from mSEED files to CSV files, (b) bulk loading data from CSV files into the DBMS (both for EagerExt), (c) loading the data directly from mSEED files into the DBMS (EagerSvr), and (d) creating primary and foreign key indexes. Additionally, Figure 2 shows the cumulative time for running all 8 workload queries both "cold" (right after restarting the server with all buffers flushed) and "hot" (with all buffers pre-loaded by running the same query multiple times after another).

The results confirm that although Lazy extracts metadata from all provided mSEED files, extracting only the metadata is orders of magnitude faster than extracting and loading all data. Also, EagerSvr is significantly faster than EagerExt, mainly due to avoiding expensive serialization to and parsing from a textual (CSV) representation. Finally, creating primary and foreign key indexes more than doubles the data ingestion times for the Eager variants. The benefit of exploiting these indexes during query processing is visible. However, the difference is rather small, such that the investment pays off only after many queries.

4.6 Querying

Figures 3 through 5 show the individual times for all 8 queries as well as their cumulative execution time ("all") for the various cases discussed above. To accommodate widely spread execution times, we use a logarithmic scale for the y-axis. We see that for cold runs, Lazy consistently outperforms both Eager variants. With hot runs, Lazy falls slightly behind Eager in some cases due to lazy ETL's on-the-fly reading of mSEED data files during execution as explained in section 3.

In Lazy, Query 3 is executed faster than Query 2 in all three datasets. This shows us that query performance of lazy ETL increases as the number of selections in the query increase, narrowing down the query. The execution times stay almost the same for eager variants, though. This is because an extra selection is introduced on the `catalog` table, although the size of result gets smaller. This also demonstrates that query performance of the lazy ETL process is dependent on how much actual data is required for the query answer. Intuitively, for the file repository case, it can range from loading no file at all (as for Query 8) to loading all the files in the worst case, where then the performance becomes similar to the initial loading of EagerSvr.

If we compare the total query times of 5000, 10000 and 20000 files, we see that query times in the eager case benefit from the indices less towards larger number of files. This is because the size of the indices is getting increasingly larger than the available main memory, which makes disk IO overhead increasingly dominant.

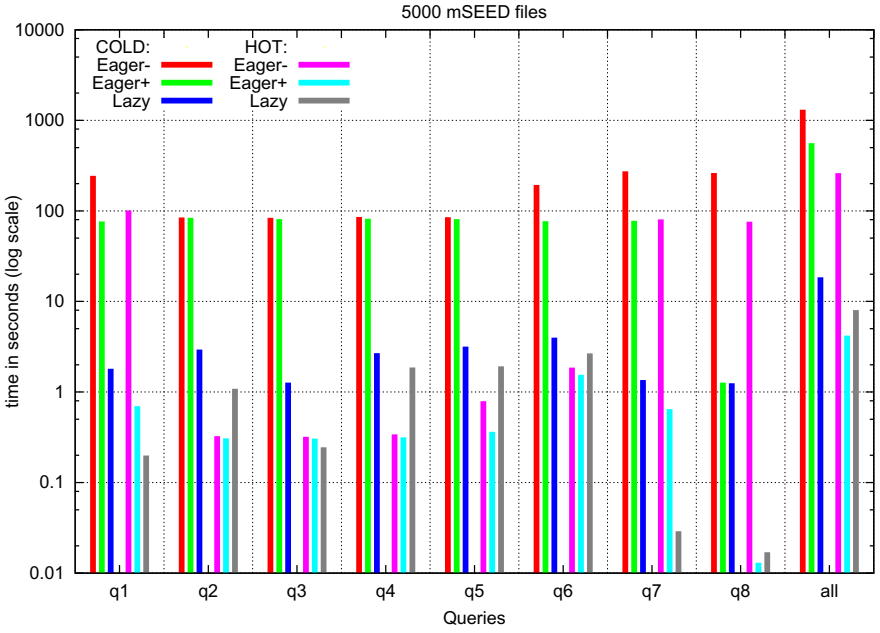


Fig. 3. Querying 5000 files

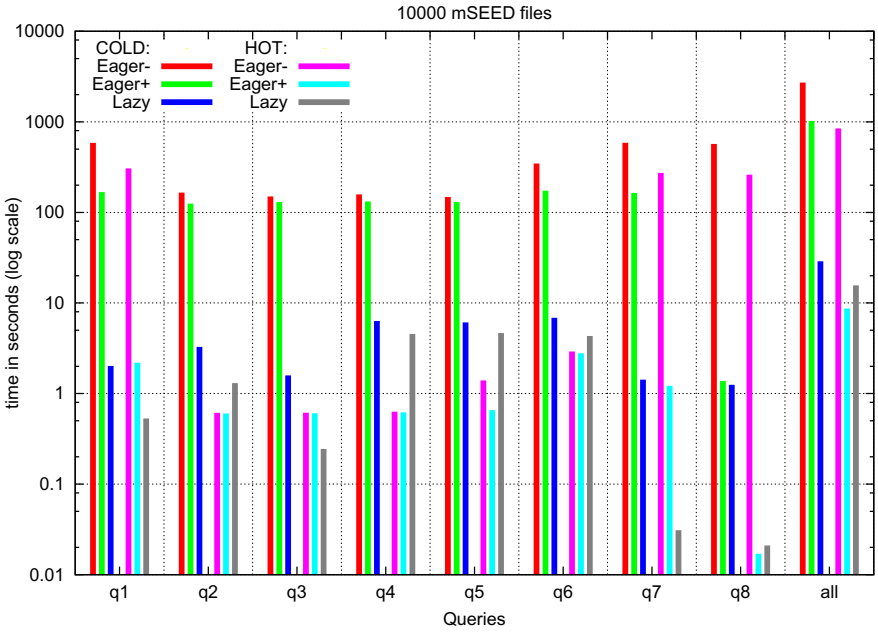


Fig. 4. Querying 10000 files

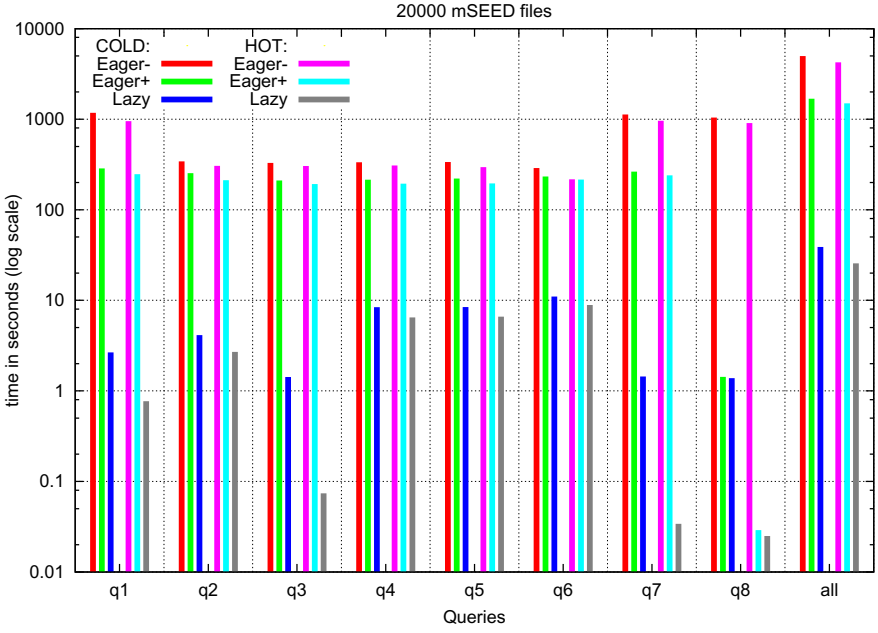


Fig. 5. Querying 20000 files

Running queries hot shows only a slight improvement in the performance for 20000 files, whereas it makes Eager approaches outperform the Lazy approach for 10000 files and even more for 5000 files. This is heavily due to the following reason. With the increase in dataset size, there is a significantly increased amount of data to deal with. Similarly, the lazy ETL process has to load more and more files and ends up with more data in the data warehouse, but still small enough to handle conveniently. This shows that lazy ETL is less vulnerable to performance problems due to dataset sizes and can more easily scale to larger data sizes than eager ETL.

4.7 Discussion

We demonstrated the significant decrease in the time from source data availability to first query answer provided by the lazy ETL. Eager ETL can lead to better query performance after spending enough effort on initial loading. However, this results in data redundancy. Our approach also provides fresh actual data, although metadata might become stale. Typical ETL techniques (i.e., periodic and incremental ETL processes) can be employed in this case for the relatively small metadata only. Moreover, since we do not actually populate the data warehouse, the amount of data to deal with for query execution in the data warehouse tends to be relatively small. Furthermore, in the resumption problem of ETL, if an eager ETL process fails, this affects the overall system performance. On the other hand, if a lazy ETL process fails, only the query that triggered it is affected.

5 Conclusion

Scientific domains can benefit from techniques developed for business ETL. We proposed in this paper that the reverse is also correct. To make it possible, we presented lazy ETL processes that integrate ETL with query processing of data warehouses. Instead of actually populating the data warehouse, we temporarily made actual data queried accessible to it. This saved us from the burden of propagating updates for most of the data into the data warehouse, which has received a lot of research effort from the ETL community. We believe this contributes to the ETL research, at least in the case where the data sources are files.

Acknowledgments. This publication was supported by the Dutch national program COMMIT. The work reported here has partly been funded by the EU-FP7-ICT project TELEIOS.

References

1. MonetDB, Column-store Pioneers, `www.monetdb.org`
2. Standard for the Exchange of Earthquake Data. Incorporated Research Institutions for Seismology (February 1988)
3. Brobst, S., Venkatesa, A.V.R.: Active Warehousing. Teradata Magazine 2(1) (1999)
4. Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. ACM Sigmod Record 26(1), 65–74 (1997)

5. Dayal, U., Castellanos, M., Simitsis, A., Wilkinson, K.: Data integration flows for business intelligence. In: Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology, pp. 1–11. ACM (2009)
6. Haas, L.M., Hentschel, M., Kossmann, D., Miller, R.J.: Schema AND data: A holistic approach to mapping, resolution and fusion in information integration. In: Laender, A.H.F., Castano, S., Dayal, U., Casati, F., de Oliveira, J.P.M. (eds.) ER 2009. LNCS, vol. 5829, pp. 27–40. Springer, Heidelberg (2009)
7. Hey, A.J.G., Tansley, S., Tolle, K.M.: The fourth paradigm: data-intensive scientific discovery. Microsoft Research Redmond, WA (2009)
8. Idreos, S., Alagiannis, I., Johnson, R., Ailamaki, A.: Here are my data files. here are my queries. where are my results? In: 5th International Conference on Innovative Data Systems Research, CIDR (2011)
9. Incorporated Research Institutions for Seismology. libmseed: The Mini-SEED Software Library (2011)
10. Inmon, B.: Operational and informational reporting. DM Review Magazine (2000)
11. Ivanova, M., Kersten, M., Manegold, S.: Data vaults: A symbiosis between database technology and scientific file repositories. In: Ailamaki, A., Bowers, S. (eds.) SSDBM 2012. LNCS, vol. 7338, pp. 485–494. Springer, Heidelberg (2012)
12. Ivanova, M., Kersten, M.L., Nes, N.J., Gonçalves, R.: An Architecture for Recycling Intermediates in a Column-store. In: SIGMOD Conference, pp. 309–320 (2009)
13. Jaeger, S., Gaudan, S., Leser, U., Rebholz-Schuhmann, D.: Integrating protein-protein interactions and text mining for protein function prediction. BMC Bioinformatics 9(suppl. 8), S2 (2008)
14. Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of data warehouses. Springer (2003)
15. Kiviniemi, J., Wolski, A., Pesonen, A., Arminen, J.: Lazy aggregates for real-time OLAP. In: Mohania, M., Tjoa, A.M. (eds.) DaWaK 1999. LNCS, vol. 1676, pp. 165–172. Springer, Heidelberg (1999)
16. Kunchithapadam, K., Zhang, W., et al.: Oracle Database Filesystem. In: SIGMOD, pp. 1149–1160 (2011)
17. Labio, W.J., Yerneni, R., Garcia-Molina, H.: Shrinking the Warehouse Update Window. In: Proceedings of SIGMOD, pp. 383–394 (1998)
18. López, J., Degraf, C., DiMatteo, T., Fu, B., Fink, E., Gibson, G.: Recipes for Baking Black Forest Databases - Building and Querying Black Hole Merger Trees from Cosmological Simulations. In: Bayard Cushing, J., French, J., Bowers, S. (eds.) SSDBM 2011. LNCS, vol. 6809, pp. 546–554. Springer, Heidelberg (2011)
19. Oldham, P., Hall, S., Burton, G.: Synthetic biology: Mapping the scientific landscape. PLoS ONE 7(4), e34368 (2012)
20. ORFEUS. Seismology Event Data (1988 - now)
21. SQL/MED. ISO/IEC 9075-9:2008 Information technology - Database languages - SQL - Part 9: Management of External Data (SQL/MED)
22. Vassiliadis, P.: A survey of extract–transform–load technology. International Journal of Data Warehousing and Mining (IJDWM) 5(3), 1–27 (2009)
23. Vassiliadis, P., Simitsis, A.: Extraction, transformation, and loading. Encyclopedia of Database Systems, 1095–1101 (2009)
24. Wetterstrand, K.A.: DNA sequencing costs: data from the NHGRI large-scale genome sequencing program (2011), `www.genome.gov/sequencingcosts` (accessed October 25, 2011) (retrieved)

Query Processing of Pre-partitioned Data Using Sandwich Operators

Stephan Baumann[1], Peter Boncz[2], and Kai-Uwe Sattler[1]

[1] Ilmenau University of Technology, Ilmenau, Germany
first.last@tu-ilmenau.de
[2] Centrum for Wiskunde, Amsterdam, Netherlands
boncz@cwi.nl

Abstract. In this paper we present the "Sandwich Operators", an elegant approach to exploit pre-sorting or pre-grouping from clustered storage schemes in operators such as `Aggregation/Grouping`, `HashJoin`, and `Sort` of a database management system. Thereby, each of these operator types is "sandwiched" by two new operators, namely `PartitionSplit` and `PartitionRestart`. `PartitionSplit` splits the input relation into its smaller independent groups on which the sandwiched operator is executed. After a group is processed, `PartitionRestart` is used to trigger the execution on the following group. Executing each of these operator types with the help of the Sandwich Operators introduces minimal overhead and does not penalize performance of the sandwiched operator, as its implementation remains unchanged. On the contrary, we show that sandwiched execution of each operator results in lower memory consumption and faster execution time. `PartitionSplit` and `PartitionRestart` replace special implementations of partitioned versions of these operators. For many groups Sandwich Operators turn blocking operators into pseudo streaming operators, resulting in faster response time for the first query results.

Keywords: indexing, clustering, partitioned data, query processing.

1 Introduction

Today, data warehouses for various reporting and analytical tasks are typically characterized by huge data volumes and a desire for interactive query response times. Over the last few years, many different techniques have been developed to address these challenges. Examples are techniques to reduce I/O by using columnar data organization schemes and/or data compression, to avoid the I/O bottleneck by keeping the data in memory (in-memory processing), and to exploit the computing power of modern hardware by using parallelization, vectorization as well as cache-conscious techniques.

Orthogonal to such improvements in raw query execution performance are existing techniques like table partitioning, table ordering and table clustering which are already found in many RDBMS products. Most of the commercial and open source databases systems support some kind of (horizontal) partitioning of tables and indexes [15,9,2]. Such partitioning is not only useful to support

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 76–92, 2013.

parallel access and to allow the query planner to prune unneeded data, but provides basically a grouping of tuples. Another related technique is clustering which also stores logically related data together. Examples are Multidimensional Clustering (MDC) in IBM DB2 [11] or partitioned B-trees [3], which are defined by distinct values in an artificial leading key column instead of a definition in the catalog. In the world of column stores, finally, storage in (multiple) ordered projections also provides a way to access data grouped by order key (range).

Though partitioning is based on a physical data organization whereas sorting and clustering are more on a logical level within the same data structure, all these techniques share a common basic concept: grouping of tuples based on some criteria like (combinations of) attribute values.

Clustering and ordering are currently most considered for indexing and exploited for accelerating selections: selection predicates on the grouping keys (or correlated with these) typically can avoid scanning large part of the data. Table partitioning also leverages this through *partition pruning*: a query planner will avoid to read data from table partitions whose data would always be excluded by a selection predicate. For joins, it holds that these are exploited primarily in table partitioning, if the partitioning key was fetched over a foreign key. In this case, for evaluating that foreign key join, only the matching partitions need to be joined. Partitioning is typically implemented by generating separate scans of different partitions, and partially replicating the query plan for each partition, which leads to a query plan blow-up. In [4] a technique is investigated to counter the ill effects of such blow-up on query optimization complexity.

In this paper we introduce a generalization of the grouping principle that underlies table partitioning, clustering and ordering, and that allows to elegantly exploit such grouping in query plans without causing any query plan blow-up. The basic idea is to not only make join operators, but also aggregation and sort operators, exploit relevant grouping present in the stream of the tuples they process. If this grouping is determined by the join, aggregation or sort key, the operator can already generate all results so far as soon as its finishes with one group, and the next group starts. Additionally, memory resources held in internal hash tables can already be released, reducing resource consumption.

The elegance of our approach is found in two key aspects. The first is the purely logical approach that treats grouping as a *tuple ordering property* captured in a synthetic `_groupID_` column, rather than physical groups, produced by separate operators. This avoids the plan blow-up problem altogether (additionally makes grouping naturally fit query optimization frameworks that exploit interesting orderings – though for space reasons, query optimization is beyond this paper's scope). The second elegant aspect are the *Sandwich Operators* we propose, that allow to exploit an ordering in join, aggregation and sort operators without need for creating specialized grouped variant implementations. This approach *sandwiches* the grouped operator between `PartitionSplit` and `PartitionRestart` operators that we introduce; and exploits the iterator model with some sideways information passing between `PartitionSplit` and `PartitionRestart`.

The remainder of this paper is structured as follows: After introducing preliminaries and basic notions in Sect. 2, we discuss the opportunities and use cases of the sandwiching scheme in Sect. 3. The new query operators implementing this sandwiching scheme are presented in Sect. 4. We implemented sandwich operators in a modified version of Vectorwise [5,17][1]. However, the general approach can easily be adopted by other systems. In Sect. 5 we discuss necessary steps and requirements and give an example. Our experiments on microbenchmarks and all 22 TPC-H queries in Sect. 6 show advantages in speed, reduced memory consumption and negligible overhead addressing the challenges of realtime data warehousing. Finally, we conclude in Sect. 8 and point out future work.

2 Preliminaries

For easier understanding we follow two definitions introduced in [16]. The first defines a physical Relation with the help of the total order $\rhd_R$

Definition 1 (Physical Relation). *A* physical relation R *is a sequence of* n *tuples* $t_1 \rhd_R t_2 \rhd_R \ldots \rhd_R t_n$*, such that "*$t_i \rhd_R t_j$*" holds for records* t_i *and* t_j*, if* t_i *immediately precedes* t_j*,* $i, j \in \{1, \ldots, n\}$*.*

In the following we will use physical relation and tuple stream or input stream interchangeable. A second definition only given informally in [16] is that of an order property, which will be sufficient for our purposes here.

Definition 2 (Order Property). *For a subset* $\{A_1, \ldots, A_n\}$ *of Attributes of a Relation* R *and* $\alpha_i \in \{O, G\}$ *(*$\alpha_i = O$ *defining an ordering,* $\alpha_i = G$ *defining a grouping), the sequence*

$$A_1^{\alpha_1} \rightarrow A_2^{\alpha_2} \rightarrow \ldots \rightarrow A_n^{\alpha_n}$$

is an attribute sequence that defines an order property for R, such that the major ordering/grouping is $A_1^{\alpha_1}$*, the secondary ordering is* $A_2^{\alpha_2}$ *and so on.*

Here, ordering (for simplicity only ascending) of an attribute A_i means that tuples of R will follow the order of the values of column A_i. Grouping of an attribute A_j is not as strong and only means that tuples with the same value for attribute A_j will be grouped together in R, but tuples may not be ordered according to values of A_j. For further reading we refer to [16].

Definition 3 (Group Identifier). *A group identifier* `_groupID_` *is an additional implicit attribute to a relation R, representing the order property* $A_1^{\alpha_1} \rightarrow A_2^{\alpha_2} \rightarrow \ldots \rightarrow A_n^{\alpha_n}$ *of R. The values of attribute* `_groupID_` *are the result of a bijective mapping function* $f : (A_1, \ldots, A_n) \rightarrow \{1, \ldots, m\}$*, where a* t_1.`_groupID_` *is smaller than* t_2.`_groupID_`*, if and only if* t_1 *precedes* t_2 *in R.*

This means, each value of `_groupID_` represents a value combination of the attributes present in the order property. In addition, `_groupID_` is reconstructable from a value combination of these attributes. Explicitly, each single occurrence of a value of an attribute is reconstructable from `_groupID_`.

[1] Vectorwise is a further development of X100 [17].

3 Motivation

Various table storage schemes result in a form of data organization where a subset of all table attributes determine an ordering or grouping of the stored data. Examples of these schemes are amongst others MDC [11] or ADC [10] or MDAM [6], where data is organized by a number of dimensions and can be retrieved in different orders. Additionally, column stores sometimes store data in (multiple, overlapping) sorted projections [14]. These methods have in common that data is not physically partitioned but has a physical ordering or grouping defined over one or multiple attributes that can be exploited during query execution. Any index scan results in a relation that contains valuable information about an ordering or grouping already present in the tuple stream. Our sandwich approach is based on having one of these forms of data organization and is designed to exploit such pre-ordering or pre-grouping. However, even systems implementing physical partitioning over one or more attributes provide the same valuable information when multiple partitions are combined into a single stream. Assuming a tuple stream that has a certain order or suborder defined over a set of attributes, we can find standard operators and show potential for optimization.

3.1 `Aggregation/Grouping`

In case of hash-based `Aggregation/Grouping`, if any subset G_s of the GROUPBY keys determines a sub-sequence $A_1^{\alpha_1} \rightarrow \ldots \rightarrow A_k^{\alpha_k}$ of the order $A_1^{\alpha_1} \rightarrow \ldots \rightarrow A_n^{\alpha_n}$ of the input tuple stream, $k \leq n$, we can flush the operator's hash table and emit results as soon as the entire group - each group is defined by `_groupID_` - is processed. Effectively, we execute the `Aggregation/Grouping` as a sequence of `Aggregation/Grouping` operators, each of which operating on only one group of data. This results in the `Aggregation/Grouping` behaving more like a non-blocking, pipelined operator, emitting results on a per group basis. Additionally, memory consumption should drop down, as the hash table only needs to be built on a subset of all keys. This may cause `Aggregation/Grouping` to no longer spill to disk, or its hash-table may become TLB or CPU cache resident. As a side effect from the reduced memory consumption we should get an improved execution time of the `Aggregation/Grouping`.

3.2 `Sort`

If a prefix $A_1^O \rightarrow \ldots \rightarrow A_k^O$ of the input relation's order $A_1^O \rightarrow \ldots \rightarrow A_n^O$ represents the same ordering as a prefix $B_1^O \rightarrow \ldots \rightarrow B_l^O$ of the requested sort order $B_1^O \rightarrow \ldots \rightarrow B_m^O$, then the tuple stream is already pre-sorted at no cost for $B_1, \ldots, B_l$ and only needs to be sorted on the remaining minor sort keys $B_{l+1}, \ldots, B_m$. This again results in executing `Sort` as a sequence of `Sort`s, each working only on a fraction of the data. The benefits of a grouped `Sort` should be similar to grouped `Aggregation/Grouping`, but additionally, as data is only sorted in small groups, the computational complexity also decreases.

3.3 HashJoin

In case of any kind of HashJoin - this also includes Semi-, Anti- and Outer-HashJoins - if a subset K_s of the join keys determines a prefix $A_1^{\alpha_1} \rightarrow \ldots \rightarrow A_k^{\alpha_k}$ of the order $A_1^{\alpha_1} \rightarrow \ldots \rightarrow A_n^{\alpha_n}$ of one input tuple stream, $k \leq n$, and K_s also determines a prefix $B_1^{\alpha_1} \rightarrow \ldots \rightarrow B_k^{\alpha_k}$ of the order $B_1^{\alpha_1} \rightarrow \ldots \rightarrow B_m^{\alpha_m}$ of the second input tuple stream, $k \leq m$, we can transform the task into multiple HashJoins, where the grouping is already present, and only matching groups induced by K_s are joined. Similar to sandwiched Aggregation/Grouping, this should result in smaller hash tables and, thus, less memory consumption for the build phase and, as a consequence from the reduced memory, better cache awareness for the build and probe phases, as well as better pipelining performance from the grouped processing. If, in addition, in both cases $\alpha_i = O, 1 \leq i \leq k$, i.e. there are only orderings involved and _groupID_ is a strictly ascending column, we can use merge techniques between the groups and skip the execution of the HashJoin for complete groups if there is no matching _groupID_.

4 Sandwich Operators

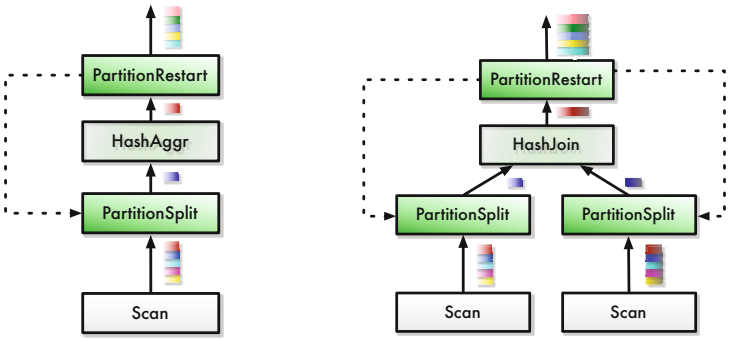
In this section we introduce two new *Sandwich* operators PartitionSplit and PartitionRestart that enable the use of (almost) unmodified existing Sort, Aggregation/Grouping and HashJoin operators to exploit partial pre-ordering of their input streams. Let this partial order be represented by an extra column called _groupID_ as introduced in 3.

The following algorithms illustrate our implementation in Vectorwise. Note that Vectorwise realizes vectorized processing of data [17], where an operator handles a vector of data at a time instead of just a tuple at a time. This enables further optimizations but the core ideas of our algorithms are transferable to tuple-at-a-time pipelining systems.

4.1 Sandwich Algorithms

Instead of implementing a partitioned variant of each physical Sort, HashJoin and Aggregation/Grouping operator, we devise a *split* and *restart* approach where the "sandwiched" operator is tricked into believing that the end-of-group is end-of-stream, but after performing its epilogue action (e.g. Aggregation/Grouping emitted all result tuples), the operator is restarted to perform more work on the next group, reusing already allocated data structures.

For this purpose we added two new query operators PartitionSplit(*stream*, *_groupID_-col*) and PartitionRestart(*stream*). The basic idea of these operators is illustrated in Fig.1 for an unary operator, HashAggr(1(a)), and a binary operator, HashJoin (1(b)). The PartitionSplit operator is inserted below Sort, HashJoin or Aggregation/Grouping and the PartitionRestart on top. PartitionSplit is used to detect group boundaries of the input stream using attribute _groupID_ and to break it up into chunks at the detected boundaries.



(a) Example Sandwich Aggregation (b) Example Sandwich HashJoin

Fig. 1. Sandwich Operators with sideways information passing

`PartitionRestart` controls the sandwiched operators restart after it finished producing tuples for a group, passes on the result tuples to the next operator and notifies its corresponding `PartitionSplit` operator(s), that the sandwiched operator is ready to process the next group. Note, that this communication between `PartitionRestart` and `PartitionSplit` is a form of sideways information passing, for which `PartitionRestart` has to know the corresponding `PartitionSplit` operator(s) in the plan. These are determined during query initialization, and typically are its grandchildren.

For both operators we outline their `Next()` methods. We also explain how Sandwich Operators are used to process the pre-grouped data of a `HashJoin` in a merge like fashion, where groups are skipped if group identifiers do not match.

```
   // initially: run=RUN nxt=veclen=redo=grp=0
 1 if this.run ≠ RUN then return 0;
 2 if ¬this.redo then
 3 |   if this.nxt = this.veclen then
 4 |   |   this.veclen ← Child.Next(); // get new tuples
 5 |   |   this.nxt ← 0;
 6 |   |   if this.veclen = 0 then
 7 |   |   |   this.run ←END; // real end-of-stream
 8 |   |   |   return 0;
 9 |   n←this.vec[this.nxt..this.veclen].SkipVal(this.grp);
10 |   this.nxt ← this.nxt + n;
11 |   if this.nxt < this.veclen then
12 |   |   this.grp ← this.vec [this.nxt ]._groupID_; // advance to next group ID
13 |   |   this.run ← STOP; // next group in sight
14 else
15 |   n ← this.redo;
16 |   this.redo ← 0 // see SkipGrp
17 return n; // return vector of n tuples
```

Algorithm 1. `PartitionSplit.Next()`

PartitionSplit() controls the amount of tuples that are passed to the sandwiched operator. When a group boundary is detected, `PartitionSplit` stops producing tuples and signals the sandwiched operator *end-of-input* while waiting for its corresponding `PartitionRestart` signal to produce tuples again.

`PartitionSplit` uses the following member variables:

`run`	- state of the operator; RUN for producing, STOP at the end of a group, END when finished with all groups.
`net`	- current position in the current vector
`redo`	- signal to produce last group once more
`veclen`	- length of current vector
`vec`	- current vector
`grp`	- current group identifier

This means that the `PartitionSplit.Next()` method, as shown in Algorithm 1, forwards tuples until a group border is detected. In line 9 `SkipVal(this. grp)` finds the number of tuples in the remaining range `[this.nxt,this.veclen]` of vector `this.grp` that belong to the same group, i.e. it finds the position where the `_groupID_` changes. On the next invocation after such a group border has been reached and all its tuples have been passed, the method has set `run=STOP` (line 13) and returns 0 (line 1), signaling (deceivingly) end-of-stream to the parent operator. This will lead an `Aggregation/Grouping` to emit aggregate result tuples of the, to this point, aggregated values (i.e. aggregates over the current group), after which it will pass 0 to its parent, in general `PartitionRestart`. In a similar way `Sort` will produce a sorted stream of the current group and `HashJoin` will either switch from building to probing or produce result tuples for the current group, depending on which `PartitionSplit` sent the end-of-stream signal. When `PartitionSplit.Next()` is called after it had previously stopped, it first checks if there are still tuples left in the current vector (line 3) and, if needed, fetches a new vector (line 4) or switches to `run=END` (line 7) if the final vector was processed.

PartitionRestart() controls the restart of the sandwiched operator and its associated `PartitionSplit`(s). In addition it applies the merge techniques in case of a sandwiched `HashJoin`.

`PartitionRestart` has the following member variables:

`Child`	- the operator below in the operator tree, usually the sandwiched operator
`lSplit`	- the corresponding (left, in case of a binary sandwich) `PartitionSplit`
`rSplit`	- in right `PartitionSplit` in case of a binary sandwiched operator

The `PartitionRestart.Next()` method (Algorithm 2) also passes on tuples (line 9) until it receives an end-of-stream signal from its `Child` (line 3). For a unary operator, it de-blocks its corresponding `PartitionSplit` if it was `STOP`ped (lines 5,6). For a binary operator it calls `PartitionRestart.GroupMergeNext()` (line 7) which handles the de-blocking of the two `PartitionSplit` operators in

```
1 n ← 0;
2 while n = 0 do
3     if (n ← this.Child.Next()) = 0 then
4         if IsUnarySandwich(this) then
5             if this.lSplit.run = END then break;
6             this.lSplit.run ← RUN; // deblock Split
7         else if ¬GroupMergeNext() then break;
8         this.Child.Restart(); // e.g., flush Aggregation/Grouping hashtable
9 return n; // return vector of n tuples
```

Algorithm 2. `PartitionRestart.Next()`

this case. Finally, it restarts the sandwiched child in line 8. If the `Partition-Split` operators do not have any more input data for the sandwiched operator, then the `while` loop is exited in either line 5 or line 7 and 0 is returned, signaling end of stream to the operators further up in the tree.

```
1 while this.lSplit.grp ≠ this.rSplit.grp do
2     if this.lSplit.grp > this.rSplit.grp then
3         if ¬this.rSplit.SkipGrp(this.lSplit.grp) then break;
4     else if this.lSplit.grp < this.rSplit.grp then
5         if ¬this.lSplit.SkipGrp(this.rSplit.grp) then break;
6 return this.lSplit.run ≠ END and this.rSplit.run ≠ END;
```

Algorithm 3. `PartitionRestart.GroupMergeNext()`

The group based Merge Join is implemented in `PartitionRestart.Group-MergeNext()` (see Algorithm 3) using a merge-join between the `PartitionSplit` operators to match groups from both input streams on its `_groupID_` values. Of course it is necessary here, that `_groupID_` is not only an identifier but also sorted ascending or descending on both sides. It is given here for `Inner-HashJoin`; for `Outer-` and `Anti-HashJoin`s it should return matching success even if one of the sides does not match (in case of an empty group).

```
1  n ← 0;
2  while this.grp < grp do
3      if this.run ≠ END then
4          this.run ←RUN; // force progress
5          n ← PartitionSplit.Next();
6      if this.run = END then return FALSE;
7      if this.vec [this.veclen − 1]._groupID_ < grp then
8          this.nxt ← this.veclen; // vector shortcut to skip search in Next()
9  this.redo ← n;
10 this.run ← RUN // Next() returns vector again
11 return TRUE;
```

Algorithm 4. `PartitionSplit.SkipGrp(grp)`

It uses `PartitionSplit.SkipGrp` (Algorithm 4) to advance over groups as long as the current `_groupID_` is still smaller than the target `_groupID_`. In turn `PartitionSplit.SkipGrp` calls `PartitionSplit.Next()` (line 5) to find the next `_groupID_` (Algorithm 1, line 12). In lines 7-8 a shortcut is used to avoid skipping over every distinct `_groupID_` in the vector (setting this.nxt to the vector length will trigger the call for the next vector in `PartitionSplit.Next()`, line 3-4). The `redo` variable used in both methods is needed, as the sandwiched operator's `Next()` call needs to receive the last tuple vector once more.

Recall that in our test implementation in Vectorwise these methods manipulate vectors rather than individual tuples, which reduces interpretation overhead and offers algorithmic optimization opportunities. For instance, the `SkipVal()` routine (not shown) uses binary search inside the vector to find the next group boundary, hence group finding cost is still linear, but with a very small coefficient. Another example is the vector shortcut in line 7 of Algorithm 4, where an entire vector gets skipped in `GroupMergeNext()` based on one comparison – checking if the last value in the vector is still too low.

We extended the (vectorized) open(), next(), close() operator API in Vectorwise with a restart() method to enable operators to run in sandwich – note that many existing database systems already have such a method (used e.g. in executing non-flattened nested query plans). This restart() method has the task of bringing an operator into its initial state; for hash-based operators it typically flushes the hash table. A workaround could be to re-initialize which may result in somewhat slower performance.

5 Application of Sandwich Operators

In order to introduce sandwich operators into query plans, the system needs to be able to generate and detect operator sandwiching opportunities.

5.1 Order Tracking and Analysis

Table partitioning, indexing, clustering and ordering schemes, can efficiently produce sorted or grouped tuple streams in a scan. Though these various approaches, and various systems implementing them, handle this in different ways, conceptually (and often practically) it is easy to add a proper `_groupID_` column to such a tuple stream. Note, that we make little assumptions on the shape of this `_groupID_` column. It does not need to be a simple integer, since our `PartitionSplit` and `PartitionRestart` operators can in fact trivially work with multi-column group keys as well. In the following, we abstract this into a scan called `GIDscan`, that a) adds some `_groupID_` column and b) produces a stream ordered on `_groupID_`

Such ordering/grouping on `_groupID_` from a `GIDscan` will propagate through the query plan as described in [16]. In our system Vectorwise, the operators `Project`, `Select` and the left (outer) side of joins preserve order and are used for order propagation.

Formally, the original ordering or grouping attributes functionally determine the `_groupID_` column. If the optimizer has metadata about functional dependencies between combinations of attributes, it will be able to infer that other groups of attributes also determine the `_groupID_` column. This order and grouping tracking and functional dependency analysis during query optimization should go hand-in-hand with tracking of foreign key joins in the query plan. The order and grouping tracking allows to identify whether aggregation and sort keys determine a `_groupID_`, providing a sandwich opportunity. The additional foreign key tracking in combination with this, allows a query optimizer to detect that the join keys on both sides on the join are determined by matching `_groupID_` columns (groups with the same boundaries), such that join results can only come from matching `_groupID_` groups on both sides of a join. This allows to identify sandwiching opportunities for joins.

5.2 Query Optimization

Sandwiched query operators consume much less memory and run faster due to better cache locality but also because its reduced memory consumption will typically eliminate the need for disk spilling, if there was one. Therefore, the query optimizer, and in particular its cost model, should be made aware of the changed cost of sandwiched operators. Note, that estimating the cost of the `PartitionSplit` and `PartitionRestart` operators is not the problem here, as they only bring linear (but low) CPU cost in terms of the amount of tuples that stream through them. If fact, thanks to the vectorized optimizations that we outlined, these costs are further reduced: (i) finding group boundaries in `PartitionSplit` uses binary search, and (ii) the merge join between groups in `PartitionRestart` typically only looks at the first and last vector values, thanks to the skip optimization. Therefore, our cost model just ignores the cost of these two operators, and focuses on adapting the cost of the sandwiched aggregation, join and sort operators. The cost model extensions are quite simple and are based on the number of groups γ_{rel} present in an input relation *rel*. `Sort` complexity decreases from $O(N \cdot log(N))$ to $O(N \cdot log(N/\gamma_{rel}))$. However, assuming a cost model that takes into account the input size of a relation, e.g. in order to calculate costs aware of a memory hierarchy, costs should be calculated as γ_{rel} times the cost for the reduced input size N/γ_{rel} (eventually not spilling anymore for hierarchy level X), as `Sort` is executed γ_{rel} times for an estimated smaller input size N/γ_{rel}. Similarly, for hash-based aggregation and `HashJoin`, one simply reduces the hash table size fed into any existing cost model (e.g. [7]) by factor γ_{rel} and multiplies the resulting costs by γ_{rel}.

As for the bigger picture in sandwiched query optimization, we note that in a multi-dimensional setup such as MDC or any partitioning, indexing, clustering or ordering scheme with a multi-column key, it may be possible to efficiently generate tuples in *many orders*: potentially for any ordering of any a subset of these keys. The potential to generate such different ordered tuple streams leads to different, and sometimes conflicting sandwiching opportunities higher up in the plan. Due to space restrictions, the question how to choose the best orderings is

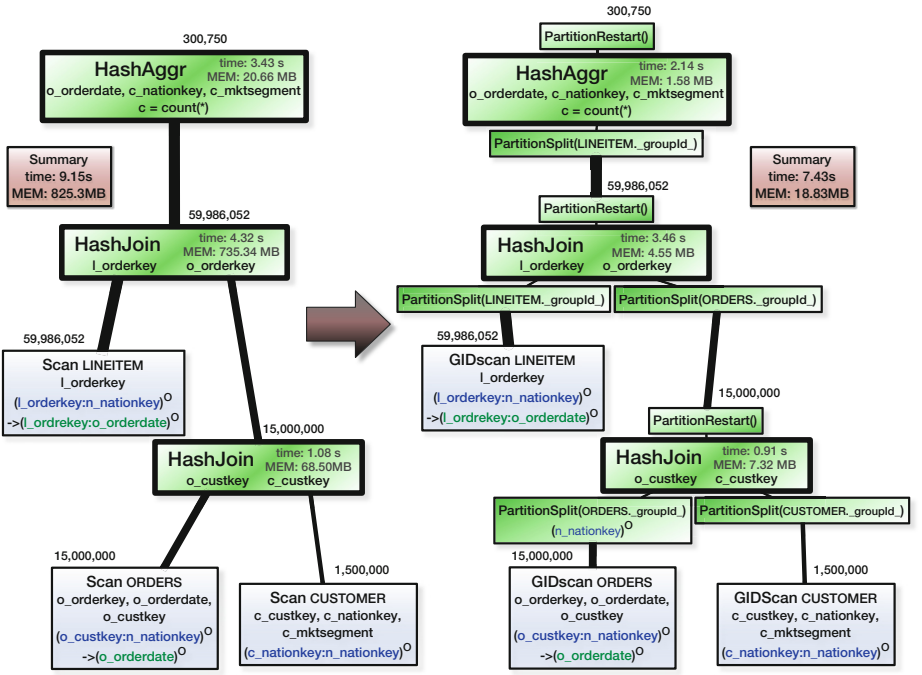


Fig. 2. Example query with and without sandwich operators

beyond the scope of this paper, but we can note here that our solution seamlessly fits into the well known concept of interesting order optimization [13], on which we will report in a subsequent paper.

5.3 An Example

In Figure 2 we demonstrate the use of sandwich operators in the following query on the TPC-H dataset:

```
SELECT o_orderdate, c_nationkey, c_mktsegment, count(*)
FROM CUSTOMER, ORDERS, LINEITEM
WHERE c_custkey = o_custkey
  AND o_orderkey = l_orderkey
GROUP BY o_orderdate, c_nationkey, c_mktsegment
```

This query is a simple version of counting the number of lineitems per market segment of each nation and date. The operators of interest are annotated with overall memory consumption and execution time, explaining the overall gain as summarized in the two summary boxes.

Assume the tables to be organized according to the following order properties:

$$\begin{aligned}
&\texttt{CUSTOMER} : \texttt{c_nationkey}^O \\
&\texttt{ORDERS} \quad : [\texttt{o_custkey}.\texttt{c_custkey}].\texttt{c_nationkey}^O \rightarrow \texttt{o_orderdate}^O \\
&\texttt{LINEITEM} : [\texttt{l_orderkey}.\texttt{o_orderkey}].[\texttt{o_custkey}.\texttt{c_custkey}].\texttt{n_nationkey}^O \\
&\qquad\qquad \rightarrow [\texttt{l_orderkey}.\texttt{o_orderkey}].\texttt{o_orderdate}^O
\end{aligned}$$

where $[A_1, A_2]$ denotes a foreign key relationship between two tables, for example `[o_custkey.c_custkey].c_nationkey`^{O} means that `ORDERS` is major sorted according to the customer nations.

As there is no information about the order of `ORDERS` or `LINEITEM` inside the nation/date groups , the original plan is still a hash based plan. Same holds for the ordering of `CUSTOMER` and `ORDERS` and ordering information about `custkey`.

However, as we have ordering properties of the tuple streams we can perform the following sandwich optimizations:

- `HashJoin(ORDERS, CUSTOMER)`: Both join keys determine the ordering on `n_nationkey`. Thus, the grouping on `CUSTOMER` can fully be exploited. Note, that `ORDERS` has a more detailed grouping, i.e. in addition to `n_nationkey` also `o_orderdate`, and for the split only the grouping on `n_nationkey` is taken into account. This is possible as we constructed `_groupID_` in a way that enabled the extraction of major orderings (see Sect. 2). In order to sandwich the `HashJoin`, `PartitionRestart` is inserted on top and one `PartitionSplit` per child is inserted on top of each input stream. `PartitionSplit` for the `ORDERS` stream needs to be provided with an extraction function of only the `n_nationkey` ordering. This results in 9x reduced memory consumption and 16% speedup.
- `HashJoin(LINEITEM, ORDERS)`: Here, both join keys determine the full ordering as given by the order properties, so the sandwich covers the complete pre-ordering. `PartitionRestart` and `PartitionSplit` are inserted similar to thx case. As this sandwich operation exploits even more groups, the memory reduction is even more significant (161x), also the speedup with 20% is higher.
- `HashAggr(o_orderdate, c_nationkey, c_mktsegment)`: Two of three grouping keys, i.e. `o_orderkey` and `c_nationkey` not only determine the ordering of the input stream but are also determined by `LINEITEM._groupID_`. That means the aggregation can be sandwiched using this pre-ordering and is only performed on a per `mktsegment` basis. For `HashAggr`, again, a `PartitionRestart` is inserted on top and a `PartitionSplit` on `LINEITEM._groupID_` is inserted on top of its input stream. Reducing the `HashAggr` to a per `mktsegment` basis accelerates the operator by 38% and reduces memory needs 13-times.

Note that in all cases, input data already arrives in a cache friendly order, i.e. the input streams are grouped in similar ways. This means that, for example, in the `ORDERS-CUSTOMER` join customers as well as orders are already grouped by nation. This results in locality for the `HashJoin` itself, an effect that is again amplified by the sandwich operators as the hash table size is shrunk.

6 Evaluation of Sandwich Operators

We evaluated on an Intel Xeon E5505 2.00 GHz with 16GB main memory, a standard 1TB WD Caviar Black hard drive for the operating system, a 64 bit

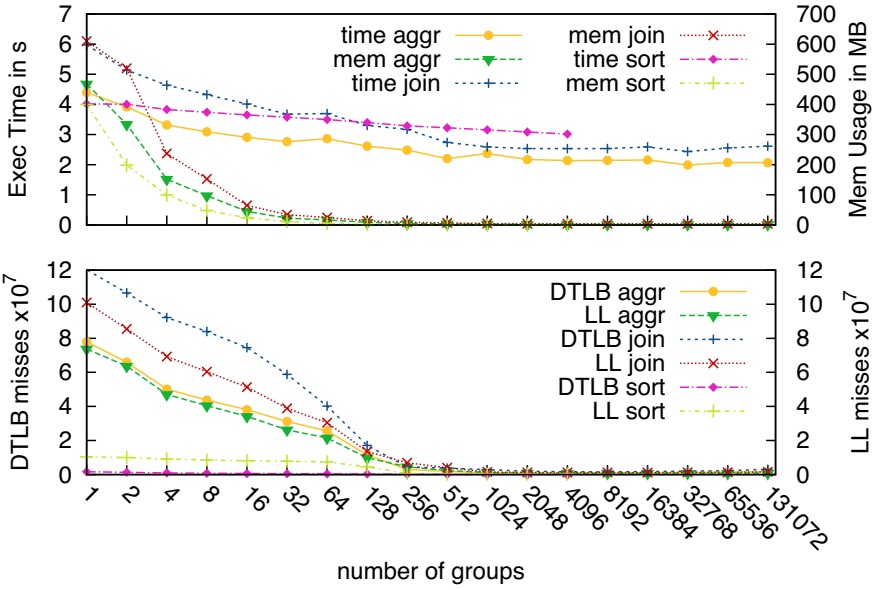


Fig. 3. Sandwiched `HashAggr` & `HashJoin`: Elapsed time, memory usage, DTLB and last level cache misses for counting the frequency of TPC-H `l_orderkey` and joining `LINEITEM` and `ORDERS` on `orderkey` using different number of groups

Debian Linux with kernel version 2.6.32. The system has 4 cores with 32KB L1, 256KB L2 and 4096KB L3 cache per core. Databases were stored on a RAID0 of 4 Intel X25M SSDs with a stripe size of 128KB (32KB chunks per disk) and a maximum bandwidth of 1GB/s. As our implementation does not yet support parallelization, queries are only executed on a single core. Our test database system is Vectorwise. It is set up to use 4GB of buffer space and 12GB of query memory. The page size was set to 32KB. The group size of consecutively stored pages was set to 1, leaving the distribution of pages to the file system.

6.1 Micro Benchmarks

Table Setup. For the micro-benchmarks for `Aggregation/Grouping` and `HashJoin` we used the `ORDERS` and `LINEITEM` table as explained in Sect. 5.3. For the `Sort` micro benchmark we used an `ORDERS` table ordered on just `o_orderdate`. Data was stored uncompressed and hot. In order to get the different number of groups, we combined two neighboring groups from one run to the next.

`Aggregation/Grouping` and `HashJoin`. The aggregation micro-benchmark scans `l_orderkey` and counts the frequency of each value. As `l_orderkey` determines the full ordering we can use the full pre-ordering for sandwiched aggregation. The join micro-benchmark performs a sandwiched hash-join of `LINEITEM` with `ORDERS` on their foreign key relationship `[l_orderkey, o_orderkey]`, with the smaller

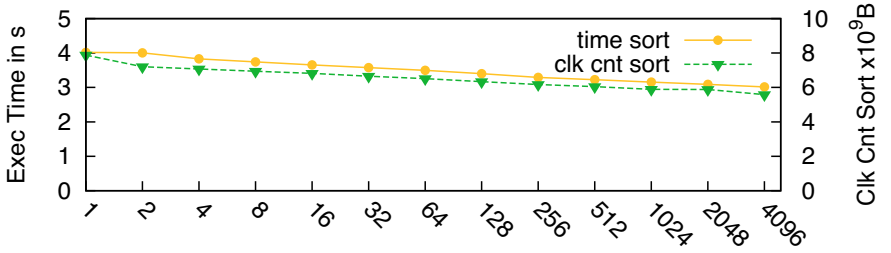


Fig. 4. Sandwiched `Sort` : Elapsed time vs. pure sorting cost for sorting `ORDERS` on `o_orderdate` using different number of groups

relation `ORDERS` as build relation. The time to scan the buffered relations is negligible, i.e. about 0.2s for `LINEITEM` and 0.05s for `ORDERS`. The same holds for memory consumption, where even in the case of 128k groups, 95% of memory is still allocated by the aggregation or join.

Their behavior is nearly identical. The upper part of Figure 3 shows that with more groups, memory consumption goes down while speed goes up. This is explained by the lower part: hash table size decreases with higher group numbers, causing the number of TLB and lowest level cache (LL) misses to drop. At 128 groups cache misses reach a minimum, as the hash table then fits into cache (15M distinct values; $15M/128 * 32B \approx 3.6MB$).

Recall that input relations may arrive in a cache friendly order and sandwich operators just amplify this cache residency effect. When comparing the `HashJoin` experiment to the example given in Section 5.3 where we can see the execution time and memory for 1 and 128k groups, it is obvious that a) the case with one group is faster and b) the case with 128k groups is slower. The explanation for a) are pipelining effects that accelerate the `LINEITEM-ORDERS` join in its build phase, as the tuple vectors from the `ORDERS-CUSTOMER` join are already in cache, accelerating the build phase by 60%. The explanation for b) is that more data is handled (two attributes from the customer relation and `o_orderdate`), which adds memory requirement and a penalty to the execution time, becoming more visible in the cache critical experiment.

`Sort.` The `Sort` micro benchmark sorts `ORDERS` on `o_orderdate` and `o_custkey` exploiting pre-ordering on `o_orderdate`. Again, neighboring groups were combined to get different granularities. The `Sort` analysis is a bit different, as cache miss numbers are between one and two orders of magnitude lower and thus there is less impact on the execution time (see Fig. 3). Detailed profiling information, however, shows that the `Sort` operator is dominated by the quick sort routines (78-82%, depending on the number of groups), and that the actual work by the CPU in these routines decreases at about the rate the execution time decreases and savings by memory access are only of minor importance (comp. Fig. 4).

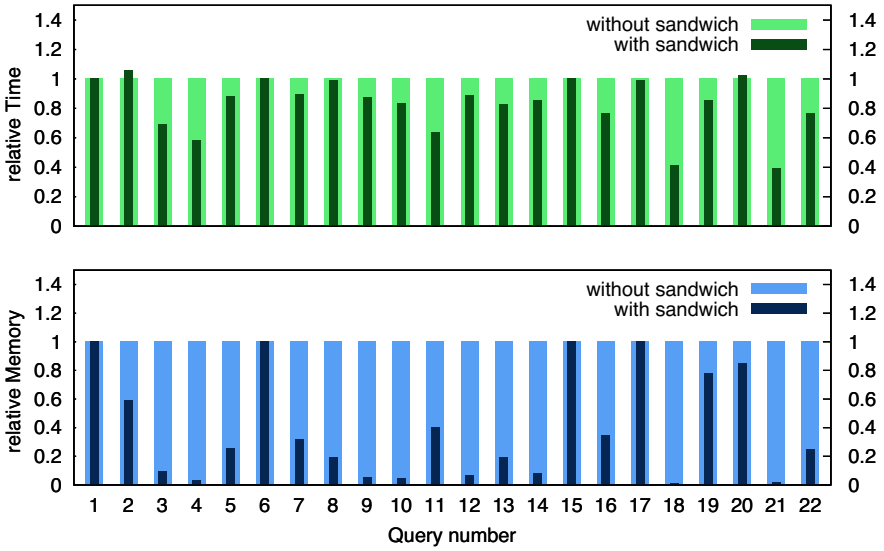


Fig. 5. Relative execution time and memory consumption for all 22 TPC-H queries with and without Sandwich Operators

6.2 TPC-H Benchmark

We prototyped our approach in Vectorwise and loaded the TPC-H SF100 data using z-ordering [8] with the dimensions `ORDERDATE`, `CUSTOMER_NATION`, `SUPPLIER_NATION` and `PART`. With this prototype we executed the 22 TPC-H queries with and without Sandwich Operators. Besides sandwiching, all other optimizations, e.g. selection pushdown to scans, were applied in both runs. This leads to Fig. 5, where we show the differences in execution time and memory consumption for both runs. Showing clear benefits for the run with the Sandwich Operators for memory consumption as well as execution time across the query set. In total Sandwich Operators saved about 129 sec (274 sec vs. 404 sec) and 22.4 GB (1788 MB vs. 24742 MB) of memory. Slight increases in execution time for Q02 (0.05sec) and Q20 (0.02sec) stem from handling the `_groupID_` in operators below the sandwiched `HashJoin`s. The partitioning benefit is not enough to make up for this as we only have very few or very small groups.

7 Related Work

In [1] special query processing techniques for MDC [11] based on block index scans are explained, that pre-process existing block-indexes before joins or aggregations and are only very briefly described. In particular their join approach fully processes the probe relation of the join to check the block index for a matching group. Without the abstraction of the `_groupID_`, this approach also requires matching attributes in order to probe the corresponding block index. Techniques

in [4] focus on processing partitioned data and, thus, have separate group matching and group processing phases and is not fully integrated in the query plan. Systems like [15,9] generate partition wise operators, where our approach reuses the same operator, saving memory and time. Work like [12] focuses on dynamic partitioning rather than exploiting orderings in relations.

8 Conclusion and Outlook

In this paper we introduced the "Sandwich Operators", an easy and elegant approach to exploit grouping or ordering properties of relations during query processing, that fits many table partitioning, indexing, clustering and ordering approaches, and though its treatment as grouping as a logical ordering property avoids plan size explosion as experienced in query optimization for partitioned tables. We showed how the sandwich operators accelerate `Aggregation/Grouping`, `HashJoin` and `Sort` and reduce their memory requirements.

As future work we see the combination the sandwich scheme with intra operator and intra query tree parallelization, where a `PartitionSplit` not only splits the input relation but distributes the groups for one or more operators among multiple cores, taking advantage of modern processor architectures.

Additionally, we left untouched the issue of query optimization for sandwiching in multi-dimensional storage schemes, where a query processor can generate tuple streams efficiently in many orders. Here, the question arises which orders to use, such that the query plan optimally profits from the sandwiching.

References

1. Bhattacharjee, B., Padmanabhan, S., Malkemus, T., Lai, T., Cranston, L., Huras, M.: Efficient query processing for multi-dimensionally clustered tables in DB2. In: VLDB (2003)
2. Chen, W.-J., Fisher, A., Lalla, A., McLauchlan, A., Agnew, D.: Database Partitioning, Table Partitioning, and MDC for DB2 9. IBM Redbooks (2007)
3. Graefe, G.: Partitioned b-trees - a user's guide. In: BTW, pp. 668–671 (2003)
4. Herodotou, H., Borisov, N., Babu, S.: Query Optimization Techniques for Partitioned Tables. In: SIGMOD (2011)
5. Inkster, D., Boncz, P., Zukowski, M.: Integration of VectorWise with Ingres. SIGMOD Record 40(3) (2011)
6. Leslie, H., Jain, R., Birdsall, D., Yaghmai, H.: Efficient Search of Multi-Dimensional B-Trees. In: VLDB (1995)
7. Manegold, S., Boncz, P., Kersten, M.: Generic Database Cost Models for Hierarchical Memory. In: VLDB (2002)
8. Markl, V.: MISTRAL: Processing Relational Queries using a Multidimensional Access Technique. Institut für Informatik der TU München (1999)
9. Morales, T.: Oracle Database VLDB and Partitioning Guide, 11g Release 1 (11.1). Oracle (July 2007)
10. O'Neil, P., O'Neil, E., Chen, X., Revilak, S.: The star schema benchmark and augmented fact table indexing. In: Nambiar, R., Poess, M. (eds.) TPCTC 2009. LNCS, vol. 5895, pp. 237–252. Springer, Heidelberg (2009)

11. Padmanabhan, S., Bhattacharjee, B., Malkemus, T., Cranston, L., Huras, M.: Multi-dimensional Clustering: A New Data Layout Scheme in DB2. In: SIGMOD (2003)
12. Polyzotis, N.: Selectivity-based Partitioning: A Divide-and-Union Paradigm for Effective Query Optimization. In: CIKM (2005)
13. Selinger, P., Astrahan, M., Chamberlin, D., Lorie, R., Price, T.: Access Path Selection in a Relational Database Management System. In: SIGMOD (1976)
14. Stonebraker, M., et al.: C-Store: A Column-Oriented DBMS. In: VLDB (2005)
15. Talmage, R.: Partitioned Table and Index Strategies Using SQL Server 2008. MSDN Library (March 2009)
16. Wang, X., Cherniack, M.: Avoiding Sorting and Grouping in Processing Queries. In: VLDB (2003)
17. Zukowski, M., Boncz, P.A., Nes, N.J., Héman, S.: MonetDB/X100 - A DBMS In The CPU Cache. IEEE Data Eng. Bull. 28(2), 17–22 (2005)

A Visual Interface for Analyzing Text Conversations

Shama Rashid, Giuseppe Carenini, and Raymond Ng

University of British Columbia, Vancouver, BC, Canada
shama.rashid@gmail.com, {carenini,rng}@cs.ubc.ca

Abstract. This paper presents a visual, intelligent interface intended to help user analyze possibly long and complex conversations. So far, the interface can only deal with synchronous conversations, but most of its components could be also applied to asynchronous conversations such as blogs and emails. In a Business Intelligent scenario, our interface could support the analysis of real-time conversation occurring for instance in blogs, discussion fora, or in sites intended to collect customer feedback. The design of our interface aims to effectively combine the power of human perceptual and cognitive skills with the ability of Natural Language Processing (NLP) techniques to mine and summarize text. Since we are targeting a large user population, with no, or minimal expertise in data analysis, we selected interface elements based on simple and common visual metaphors. Furthermore, to accommodate for user differences in such a large population, we provided users with the ability to satisfy the same information needs in different ways.

We have tested our interface in a formative user study, in which participants used the interface to analyze four long and complex conversations, in order to answer questions addressing specific information needs in a business scenario. This evaluation revealed that the interface is intuitive, easy to use and provides the tools necessary for the task. Participants also found all the interface components quite useful, with the main problems coming from inaccuracies in the information extraction process, as well as from deficiencies in the generated summaries. Finally, it seems that the choice of offering redundant functionalities was beneficial. The logged interaction behaviors reveal that different users actually selected very different strategies, even independently from their performance. And this was consistent with the high variability in the user preferences for the different interface components.

Keywords: interacive interface, multi-modal interface, ontology, conversation visualization, conversation browsing and summarization.

1 Introduction

In our daily lives, we have conversations with other people in many different modalities. We email for business and personal purposes, attend meetings in person and remotely, chat online, and participate in blog or forum discussions.

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 93–108, 2013.

The Web has significantly increased the volume and the complexity of the conversational data generated through our day to day communication and has provided us with a rich source of readily available private and public discourse data.

It is clear that automatic summarization can be of benefit in dealing with this overwhelming amount of information by providing quick access to possibly large and complex conversations that are constantly evolving in real-time. Automatic meeting summary would allow us to prepare for an upcoming meeting or review the decisions of a previous group. Email summaries would aid corporate memory and provide efficient indices into large mail folders. Summaries of blogs could help people join large ongoing conversations in real-time and support them in searching and browsing particular sub-topics and discussions.

Previous work on summarizing conversations, however, indicates that summarization techniques generating generic, exclusively textual summary are often insufficient [11]. In contrast, on the one hand, [7] shows that meeting summaries focused on user's information needs are more effective than generic ones, even if they contain several inaccuracies. On the other hand, [18] demonstrates that visualization and flexible interactive techniques can compensate for the limitations of state-of-the-art summarization methods, especially when applied to complex text mining tasks.

Building on these findings, in this paper we present a novel interface that takes advantage of human perceptual and cognitive skills in conjunction with automatically extracted information and summarization capabilities to support users in analyzing a conversation, to satisfy a set of related information needs. The automatically extracted information includes an annotation of each utterance in the conversation for what entities it is referring to and also for the dialog acts it is expressing, such as whether it is conveying a positive vs. negative subjective orientation, a decision, a problem, or an action item.

In addition to aiming for an effective symbiosis between human perceptual and cognitive skills and the computer ability to mine and summarize text, the design of our interface follows three basic principles.

- Simplicity: the interface components are based on common visual metaphors (facets [21] and word clouds [1]), so that the interface can be used by a large user population.
- Redundancy: most common information needs can be satisfied in multiple ways, by performing different sequences of visual and interactive actions. The goal is to accommodate for, as well as to investigate, user differences in preference and expertise, that we expect to be present in such a large population.
- Generality: although we have only applied our interface to meeting transcripts, most of the underlying algorithms for information extraction and the information visualization techniques are not meeting specific and would work on different conversational modalities (e.g., emails, blogs).

We have tested our interface in a formative user study, in which participants used the interface to analyze four long and complex conversations, in order to answer questions addressing specific information needs. This evaluation revealed that

the interface is intuitive, easy to use and provides the tools necessary for the task. Participants also found all the interface components quite useful, with the main problems coming from inaccuracies in the information extraction process, and from deficiencies in the generated summaries. In terms of interactive behavior, despite a rather substantial training, participants exhibited a broad range of strategies, with no clear cut distinction between top and bottom performers.

As a preview, in the remainder of the paper, we first describe the NLP techniques for information extraction and summarization underlying our approach. Next, we present the design and implementation of the interface. After that, we discuss related work, and conclude by presenting the user study and discussing the key findings.

2 NLP: Information Extraction and Focused Summarization

Our browsing and summarization methods rely on mapping the sentences in a conversation to an ontology containing three core upper-level classes for participants (i.e., speakers), dialogue acts (DAs) (e.g., decision), and referred entities (e.g., battery). An example for mapping a sentence to the ontology is shown in Figure 1.

A: *Let's go with a simple chip.*
Speaker: A, who is the Project Manager
Entities: simple chip (only one for this particular example)
Dialog Acts: classified as decision and positive-subjective

Fig. 1. Example of mapping a sentence to an ontology

We represent our ontology in OWL/RDF (Web Ontology Language/Resource Description Framework) [11]. The mapping shown in Figure 1 would be expressed in OWL/RDF as follows:

```
<Utterance rdf:about="#TS3012a.A.dialog-act.vkaraisk.14">
   <rdf:type rdf:resource="&owl;Thing"/>
   <hasSpeaker rdf:resource="#ProjectManager"/>
   <hasDAType rdf:resource="#Decision"/>
   <hasDAType rdf:resource="#PositiveSubjective"/>
   <begTime>18.61</begTime>
   <endTime>20.49</endTime>
</Utterance>
```

In term of processing, the Participant annotations for the sentences are extracted directly from the transcript of the conversation. As for the Entity class, it contains noun phrases referred to in the conversation with mid-range (10%-90%) document frequency after filtering for non-content words and stop words (words like 'anyone', 'okay' etc.). This selection strategy avoids considering overly general/specific terms. The DA annotations for sentences are determined using supervised classifiers. Our classifiers are designed for identifying five subclasses of the DA-type class, namely action items, decisions, negative and positive subjective, and problems. However, we could easily include additional classifiers to identify other types of DAs to support a larger variety of the information needs. The classifiers rely on a feature set related to generic conversational structure, including features like sentence position in the conversation and in the current turn, pause-style features, lexical cohesion, centroid scores, and features that measure how terms cluster between conversation participants and conversation turns. The classifiers also use sentence level features like word pairs, Part of Speech (POS) pairs, character trigrams etc. Overall the classification accuracy is quite high ranging from .92 to .76 (AUROC metric). Notice that these classifiers do not rely on any feature specific to conversations in a particular modality (e.g., meetings), so they can be applied to conversations in other modalities and even to multi-modal conversations (e.g., a conversation started in a meeting and continued via email). This follows our design principle of generality.

The automatic mapping of the sentences into the ontology is the key knowledge source for generating focused summaries that satisfy some given user information needs. These summaries can be either extractive or abstractive (see redundancy principle). An extractive summary is simply generated by extracting a subset of the sentences in the conversation satisfying those needs, while an abstractive summary is generated by extracting and aggregating information satisfying those needs and then generating new sentences expressing the selected information.

We generate abstractive summaries by following the approach originally proposed in [10,11]. First, sentences are aggregated into messages based on whether, for instance, they are uttered by the same participant and express the same dialog act on the same entity. Then, the most informative messages are selected by using an optimization function combining sentences and messages subject to three constraints: a summary length constraint and two constraints tying the information value of messages and sentences together. A Natural Language Generation component is finally used to produce summary sentences from the selected messages.

3 Interface Design

3.1 Display Design

As shown in Figure 2 our visual interface consists of four integrated views, the Ontology View (upper left), the Entity View (bottom left), the Transcript View

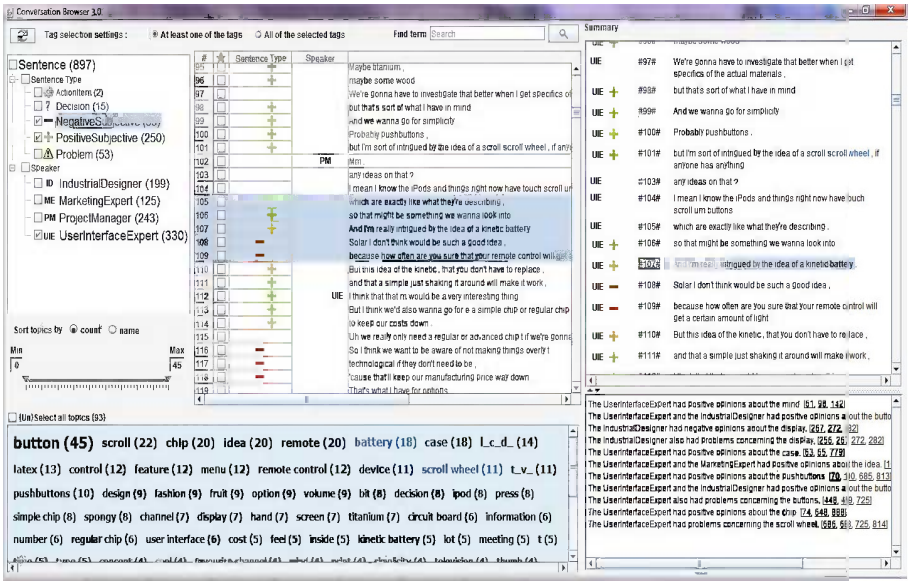


Fig. 2. A visual interface for analyzing conversations with 4 integrated views: a) the Ontology View (top left), b) the Transcript View (middle), c) the Entity View (bottom left), and d) the Summary View (right)

(middle) and the Summary View (right). They all contribute, in an often redundant way, to support the exploration and analysis of the conversation through its mapping of sentences into an ontology, as well as through the ability of generating focused summaries that cover only aspects of the conversation that are especially relevant to the user's information needs.

The Ontology View. The Ontology View provides a systematic way for the users to explore the relevant concepts in the conversation and their relations. It contains a faceted tree hierarchy with core nodes Speaker and DA-type. The root node in the ontology tree represents all the sentences in the conversation, while any other node represents a subset or subclass of those sentences that satisfy a particular property. For instance, the node ProjectManager (PM) represented all the sentences uttered by the PM, while the node ActionItem represented all the utterances that were classified as containing an action item. For leaf nodes, the counts beside the labels act as information scent and indicate how many sentences are mapped to this node. For instance, in the Speaker subtree these counts provide a sense of how dominant a speaker was in this particular conversation. Also, notice in Figure 2 that leaf labels are scaled according to their count, to make more frequent nodes stand out more.

The user can select multiple nodes on the Ontology View tree using checkboxes juxtaposed to the node labels. To take advantage of visual popout, we

have associated an icon with each of the ontology concepts. Following domain convention, we are using a green '+' or a red '-' shaped icon for PositiveSubjective and NegativeSubjective nodes respectively. For the other DA-type nodes, we have used the shape of the most common icon found when we googled using the keywords. For speaker nodes, abbreviations of the speaker names are used. For instance, the icon for the Industrial Designer is ID.

The Entity View. The Entity View is a textual collage of the list of entities mentioned in the conversation represented as a tag cloud (with actual counts in parenthesis). This view not only represents a quick overview of the content of the whole conversation, but also provides access points into the conversation. The user can search the conversation for sentences mentioning any subset of the displayed entities by simply selecting that subset.

The Transcript View. The Transcript View shows the sentences of the conversation one per row, ordered temporally in the 'Sentence' column. Gridlines are included to make the separation of the sentences apparent. However, sentences belonging to a turn by the same speaker are grouped using containment within a larger grid box. Notice that if an entity has been selected in the Entity View, it will be highlighted in the transcript. Moving now to the left of the sentence column (see Figure 2), additional information is shown in auxiliary columns. The two columns,'Sentence Type' and 'Speaker', show, for each sentence, the icons for the DA-type and Speaker to which the sentence was mapped. The next column to the left (with header star icon) allows the user to mark a sentence as important for her current information needs, while the leftmost column numbers utterances in the order in which they were uttered.

The Summary View. The Summary View comprises two panels. The top panel displays a focused extractive summary, while the bottom one displays a focused abstractive summary. The extractive summary includes all the utterances that were mapped to concepts on the Ontology View and the Entity View currently selected by the user. Notice that the user can choose whether she wants to include utterances mapped in at least one of the concepts vs. utterances mapped to all the selected concepts. For instance in Figure 2, the user made the former choice, so the extractive summary includes all the utterances that are either subjective (positive or negative), or uttered by the Industrial Designer, or containing at least one of the selected entities (i.e., battery and scroll wheel). While the extractive summary is simply a subset of the utterances in the conversation transcript, the abstractive summary in the bottom panel comprises sentences that are generated from scratch as described in the Information Extraction and Focused Summarization Section. The abstract sentences summarize all the utterances that in the corresponding extractive summary are simply included verbatim. Finally, as shown in Figure 2 (right bottom), each abstractive sentence is followed by a list of line numbers that link the abstractive sentence to conversation utterances the sentence is summarizing.

3.2 Interactive Techniques

When the Transcript View is generated for a conversation, the 'Sentence Type' column is initially empty and all the nodes on the ontology tree in the Ontology View are shown fully expanded and are de-selected. The user can minimize(/expand) nodes she is-not(/is) interested in on this tree hierarchy.

Once the user selects a node (or de-selects an already selected node) on the ontology tree, the keyword or icon associated with that node appears in (or disappears from) the 'Sentence Type' column of all the rows that contain sentences that can be mapped to that particular node; in case of Speaker nodes, the icons are shown all the time to keep the user oriented to who is saying what. However, selecting a Speaker type node on the Ontology View changes the filtering criteria for the summary. Once the user has selected the nodes of interest from the ontology tree and the Entity View tag cloud, she can scroll through the Transcript View and inspect sentences that appear to be promising to satisfy her information needs.

The Tag Selection Settings control (top left) allows the user to choose whether she wants to include utterances mapped in at least one of the selected concepts vs. utterances mapped to all the selected concepts.

A Range Slider is provided for the Entity View (on the top left corner of the view) to narrow down the set of displayed entities based on how many times they occurred in the conversation. Entities outside the selected range fade out.

The Entity Sort Order control (on top of the Range Slider) can be used to display the entities in the Entity View in alphabetical order or according to their frequency counts.

The Summary View is linked to the Transcript View. When the user clicks on a sentence in the extractive summary or on a link in the abstractive summary, the corresponding sentence in the Transcript View is highlighted, along with the two preceding sentences and the two subsequent sentences to make the highlight easier to spot. Simultaneously, the viewport on the Transcript View is adjusted to show the highlighting by auto-scrolling.

Finally, the interface provides a standard Keyword Search Box (top middle) and a Reset button. The Search Box allows the user to inspect the sentences in the transcript referring to a search term, one at a time, until the end of the transcript has been reached. The Reset Interface button at the top left corner of the interface allows the user to deselect all currently selected nodes for speakers, DAs and entities.

3.3 Implementation

Our prototype has been developed iteratively, using Java Swing and AWT components and Jena, an open source Java framework that provides a programmatic environment for building semantic web applications. The NLP techniques for information extraction and summarization were implemented in Python. We have also used Python scripts to do the data processing at the back end of the interface.

4 Related Work

Our approach to support the exploration of conversations by explicitly showing a mapping of all the utterances in an ontology is an example of the highly successful and popular faceted interface model [13]. Flamenco [21] and Mambo [6] are two influential systems based on facets. They make use of hierarchical faceted metadata for browsing through image or music collections, respectively. As it is commonly done in faceted interfaces, we have included a count beside each node of the ontology to indicate the number of utterances in the conversation that have been mapped to it.

The distribution of terms in a text document has been explored using different techniques (e.g., tag cloud [1], TextPool [3], Wordles [17]) ; the central theme of all of them being the use of size to encode the frequency of occurrence of a term. We have adopted a tag cloud format to list the entities in our interface, because of its simplicity and popularity.

Several aspects of our approach have been investigated in previous work on supporting the analysis of meeting conversations. The Ferret Meeting Browser [19] allows navigation by clicking on a vertical scrollable timeline of the transcript. More recently, the Meeting Miner [2] enhances timeline-based navigation, with the possibility of retrieving a set of speech turns by specifying keywords that can be typed in or selected from a list of automatically generated keywords and topics. Similarly, we provide these functionality in the Entity View and in the Keyword Search Box.

The summarization system presented in [7] makes use of dialog acts for generating an extractive decision-focused summary suitable for debriefing tasks for conversations about designing a new product (see AMI corpus [4]); our approach also considers the speaker, subjectivity and entity information. Furthermore, it provides flexibility in choosing the summarization technique (extractive vs. abstractive). Another key difference is that our information extraction methods only use textual and conversational features and are therefore applicable to conversations in different modalities (e.g., emails). In contrast, the system proposed in [7] is meeting specific, because it takes advantage of the audio-video recordings to better understand decision points.

The recent CALO meeting assistant [15] is also quite similar to ours. However, while they do perform an arguably more refined semantic analysis on the transcript, in term of topic identification and segmentation, question-answer pair identification, as well as the detection of addressees, action items and decisions, they only provide extractive summaries of the conversation. More tellingly, their interface does not appear to be as visually sophisticated as ours, as it does not exploit the power on faceted search and tag clouds. One interesting feature of the CALO assistant is that it allows users to attach their own annotations to the transcripts, which is an interesting direction we could explore in our future prototypes.

5 Evaluation

Generally speaking, interfaces can be empirically tested with users in two ways [14]. In a comparison evaluation two interfaces are tested to see which one is superior with respect to a set of usability and performance metrics. In an assessment evaluation, only one interface is tested to verify properties of the interface, including: whether the user tasks are effectively supported, whether there are usability problems, and to identify relevant user differences in term of preferences and behaviors. In this paper, we present an assessment evaluation of our interface. In the user study, participants were asked to use the interface to browse and analyze a set of four long and complex human conversations (from the AMI corpus [4]). In particular, participants were asked to answer specific questions about the discussions that took place during the length of the conversations.

Thirty participants were recruited and compensated for the approximately 2 hours they spent on the study. A prize for the top three scorers was also offered to encourage people to get engaged in the task assigned. Two of the participants were excluded from the final analysis since they scored well below the average (more than two standard deviations). The remaining twenty-eight subjects ranged in age from 20 to 32. Twenty were male and eight were female.

We asked the participants to self-assess their comfort level using computers on a scale ranging 1 to 10 (where 1 meant they rarely used computers and 10 meant they could be considered expert computer users) and the median value was 8.5 out of a range of values from 3 to 10. All of the participants had normal or corrected vision.

The study procedure comprises three sessions - (a) a tutorial, (b) a practice and (c) an experiment session. The first two sessions were designed to ensure that the participant had acquired a common working understanding of the information displayed and of the interactive tools, before moving on to the experiment session. The instructions provided to the participants were carefully scripted

Table 1. Study sessions, sample tasks and sample marking scheme for judges

Study Session	**Sample Tasks**	
Tutorial 1 conversation 2 participants 50 sentences	2. What kind of coat does Susan want to buy? Why did she not want a red coat or a tri-climate one? 3. What does Susan want for lunch? Did she consider other options? If so, what were the other choices?	
Practice 1 conversation 2 participants 100 sentences	1. What types of movies did Betty and Ronnie consider watching? What movie did they finally decide to watch? Why did they reject the other options? 2. What party are they planning to go to? Why is Betty worried about preparing for the party?	
Experiment		**Judges' Marking Scheme**
4 conversations 4 participants # of sentences min 363 max 1401 avg 901	2. A recent market survey revealed that battery life was an important feature considered by customers while buying a remote control. What are the options the design group considered about power of the remote? What was the final decision? Who proposed that solution?	Full marks: 3 4 options (0.5 marks each): a) double A, b) solar power, c) Lithium ion or long lasting or rechargeable and d) kinetic batteries; final choice kinetic battery (0.5 marks); proposed by the User Interface Expert (0.5 marks)
	3. One of the online customer reviews for the product read "A scroll-wheel like Apples ipod would have been cool for channel surfing!" Did the project group consider this option? If so, why was it discarded? What did they decide on in the end as the main way of interaction?	Full marks: 3 Yes, they discussed the scroll wheel (1 mark); decided to use push buttons instead (1 mark); rejected the scroll wheel because it is a) expensive and b) harder to control (1 mark if either reason mentioned)

for all three sessions to ensure that every participant was provided the same overview of the interface and how the different components can be used to derive the answers. At the beginning of the user study, we asked the users to fill up a pre-questionnaire to gather some background information.

In the tutorial session, the experimenter explained the different components of the interface to the user using simple tasks (see Table 1 for examples) on a short conversation between two participants. The user was advised to take some time to get comfortable using the interface at the end of this session. Next, in the practice session, the user worked on a slightly longer conversation of 100 sentences between two participants. The user was assigned two tasks (see Table 1) and was encouraged to work on her own. At the end of this session, the experimenter provided the user feedback on her performance for the assigned tasks.

The experiment session was timed with a limit of 1 hour. The user was assigned five tasks in a business oriented scenario (see Table 1 for two samples) based on a series of four related meeting conversations (ES2008 series of the AMI corpus) on the design of a new remote control by a group of four people participating with very specific roles. The four conversations were displayed on separate tabs, each tab containing all the basic controls and views as show in Figure 2. The settings and controls on one tab worked independently of the controls and settings on other tabs. The conversations ranged in length from a few hundred to more than a thousand sentences (see Table 1 for details). The five tasks in the experiment session were designed so that they could be answered independently of each other and in any order the user preferred.

At the end of the experiment we administered a questionnaire to get feedback on the usability of the interface, usefulness of its components and to get suggestions to improve the interface. The questionnaire consisted of Likert scale questions with scale value ranging from 1 to 5 and some open ended questions on different aspects of the interface.

All the sessions were carried out on a standard Windows 7 machine with 6 GB RAM, a 23 inch monitor and standard keyboard and mouse devices. During the experiment, interaction behaviors were automatically logged.

The answers of each participant were scored for accuracy by two human judges not associated with the project. Two sample marking schemes are shown in Table 1, besides the corresponding experimental tasks. The independent sets of scores from the two judges were highly correlated (Pearson coefficient = 0.89). The average of the two scores was used in the final analysis.

6 Results

The aim of the user study was to assess the usability and effectiveness of our interface from different angles. Furthermore, since user accuracy on the task can be measured quantitatively, we are able to cluster users based on their performance and verify whether specific behavioral patterns and/or preferences for certain interface components are related to performing better or worse on

the assigned tasks. For instance, we can answer questions like: Is it the case that the best performers used the Summary View more frequently? Or, is it true that the worst performers preferred more the Search Box. Answering these and similar questions on user behavior and preferences could inform the redesign of the interface and improve the training of new users.

6.1 Participants' Scores

As shown in Table 2, the performance of the participants varied substantially. We had three participants who achieved a perfect score of 12. To investigate user differences related to performance, we cluster our twenty eight participants into 3 mutually exclusive groups based on the scores they achieved: a high performer group containing nine people with performance score greater than 11, a mid-performer group containing ten people with performance score in the 8.5-11 interval, and a low performer group containing nine people with performance score within the 5.75-8.5 interval.

Table 2. Summary statistics for scores of the 28 participants

Min	Max	Mean	Median	Std Dev
5.75	12.00	8.87	9.50	2.83

6.2 Time Performance

Most participants took all the time allotted for the tasks. The ones who did not, finished only a few minutes earlier and did not specifically belong to any of the three groups based on the scores. Thus, we do not discuss time performance any longer.

6.3 Pre-questionnaire

In the pre-questionnaire, we gathered information on the participants' gender, education level, comfort level in English, computer proficiency or familiarity. None of these properties showed a statistically significant correlation (at 0.05 significance level) with performance.

6.4 Post-questionnaire

The questionnaire at the end of the experiment was designed to collect feedback on the overall usability of the interface, the utility of the different components and to gather suggestions on possible improvements. Table 5 shows the questions on the interface usability and perception of task context. Participants expressed their answers on a Likert Scale in which Strongly agree = 5; Strongly disagree = 1. In general, users found the interface intuitive, easy to use, and providing the necessary tools (median vales for Q1 and Q7 was 4). The participants also

reported that they felt they were able to find relevant information quickly and efficiently (median value 4 for Q3), but sometime not all the needed information was available (median value 3 for Q2).

The fact that all the questions about the perceived task context had median value equal to 3 is an indication that the tasks were at an appropriate level of difficulty, not too easy, nor too difficult.

The post-questionnaire also gathered more specific feedback about the usefulness of the main components of the interface, and about the perceived accuracy of the information extracted by the NLP techniques. Results on these two aspects are shown in Table 3 and Table 4, respectively. From Table 3, it appears that, overall, a large majority of the participants found the three main components to be at least somewhat useful, with the Entity View being the most useful one. Presumably, the Ontology View was rated by most user as somewhat useful or not useful, because as shown in Table 4 participants found the information displayed in the Ontology View (i.e., DA tagging) less accurate than the one in the Entity View (i.e., mid-frequency noun phrases). Based on this observation, future work should be focused on improving the accuracy of our classifiers.

With respect to differences among the three groups of participants based on performance, we did not notice any interesting difference or trend in either Table 3 or 4.

A final remarkable result from the post-questionnaire is that most participants did not find the abstractive summary very useful. Three participants explicitly mentioned that the abstractive summary is not well organized or useful, and needs more content. Others blamed lack of context for abstractive summaries being unsuitable as a browsing tool. In practice, only four participants tried out using the abstractive summary as a means of navigation and they clicked on less than 10 links each. This contrast with the results found in [11] in which

Table 3. User feedback on the main components of the interface. Legend: + indicates Useful, +/- Somewhat Useful, - Not Useful. (Some overall totals do not sum up to 28 because of missing data from the questionnaire.)

Component	Overall			High Perf.			Mid Perf.			Low Perf.		
	+	+/-	-	+	+/-	-	+	+/-	-	+	+/-	-
Ontology View	9	14	5	2	5	2	3	5	2	4	4	1
Entity View	19	5	3	8	0	1	7	2	0	4	3	2
Summary View	14	10	3	6	3	0	2	5	3	6	2	0

Table 4. User feedback on the perceived accuracy of the two main Information Extraction tasks. Legend: + indicates Accurate, +/- Somewhat Accurate, - Inaccurate. (Some overall totals do not sum up to 28 because of missing data from the questionnaire.)

Component	Overall			High Perf.			Mid Perf.			Low Perf.		
	+	+/-	-	+	+/-	-	+	+/-	-	+	+/-	-
Listed Entities	15	11	0	6	3	0	4	4	0	5	4	0
DA Tagging	13	6	8	4	3	2	5	3	2	4	0	4

Table 5. Post-questionnaire questions. Assessments were expressed on a Likert Scale in which Strongly agree = 5; Strongly disagree = 1. For questions preceded by '*' lower values are better

Questions	Mean	Median
Interface Usability		
Q1 I found the conversation browser intuitive and easy to use.	3.714	4
Q2 I was able to find all of the information I needed.	3.321	3
Q3 I was able to find the relevant information quickly and efficiently.	3.393	4
Q7 I had the necessary tools to complete the task efficiently.	3.571	4
Q8 I would have liked the conversation browser to have contained additional information about the conversations.	2.750	3
Q9 The interface quickly reflected the changes caused by interaction (changes caused when you select or unselect tags etc.)	4.036	4
Task Performance		
Q4 I feel that I completed the task in its entirety.	3.214	3
Q5 The task required a great deal of effort.	3.250	3
Q6 I felt I was working under pressure.	2.857	3

abstractive summaries were assessed to be superior than extractive one. However, that study was quite different from ours. In [11], generic summaries were assessed based on their form and content, in our study they were assessed in the context of a realistic information seeking task.

6.5 Behavioral Patterns

The usage of different components of the interface was automatically logged based on clicking and scrolling behaviors. We were interested in answering the following key question: Do users, who perform better at the assigned tasks, follow different strategies (in term of components usage), than users who perform worse?

In general, our analysis of the interaction data indicates that high performers did not apply similar strategies. And the same is true for low performers. On the contrary, while some participants relied heavily on a single interaction techniques or a single view to explore the conversations, other adopted a more balanced strategy, independently from their performance level. For instance, even among the participants who achieve perfect score, one relied heavily on the generic Keyword Search (see participant P19 in Figure 3), while another one used the interface tools and views in similar proportions. For an additional example, look at Figure 4, which shows how two participants (P27 and P17), from two different performance groups relied heavily on the same interactive technique, namely the link between the extractive summary and the transcript for navigation.

These findings appear to support the design choice of offering redundant functionalities within the interface. Different users seem to be able to achieve top performance in different ways, depending on their skills and preferences.

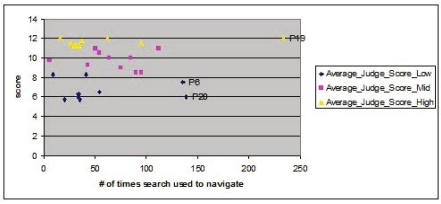


Fig. 3. Scatter plot for the number of times search button was used to navigate by a participant and participant's score. Participant 19 is one of the perfect scorers.

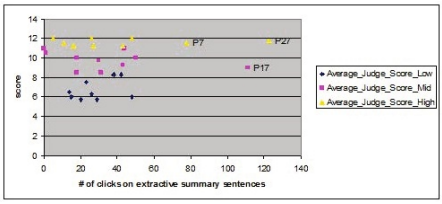


Fig. 4. Scatter plot for the number of time participants clicked on the extractive summary sentences and their score. P27, one of the high performers, and P17, one of the mid-level performers, both relied heavily on extractive summary for navigating within the transcript.

7 Conclusions and Future Work

This paper presents a visual, intelligent interface intended to help user analyze possibly long and complex conversations. The design of our interface aims to effectively combine the power of human perceptual and cognitive skills with the ability of NLP techniques to mine and summarize text. Since we are targeting a large user population, with no, or minimal expertise in data analysis, we selected interface elements based on simple and common visual metaphors. Furthermore, to accommodate for user differences in such a large population, we provided users with the ability to satisfy the same information needs in different ways.

A comprehensive formative evaluation of the interface indicates that while we were quite successful in our endeavor, some problematic aspects still need to be addressed. On the positive side, not only users found the interface intuitive, easy to use and providing the necessary tools, but they also considered most of the key interface components to be useful. Furthermore, it seems that the choice of offering redundant functionalities was beneficial. The logged interaction behaviors reveal that different users actually selected very different strategies, even independently from their performance. And this was consistent with the high variability in the user preferences for the different interface components.

On the negative side, it seems that the NLP techniques generating the information displayed in the interface are still somehow unsatisfactory. More specifically, half of the users were at least partially bothered by inaccuracies in the classifiers that map utterances into the DAs they express. Furthermore, most

users did not find abstractive summaries particularly useful. And abstractive summaries essentially represent the most sophisticated NLP component of our framework.

Based on these findings, our goal in the short term is to improve the NLP techniques. For the classifiers, higher accuracy could be achieved by applying more sophisticated machine learning techniques [5] and/or exploiting more informative sentence features (e.g., from a dependency parser). For the abstractive summarizer, improvements should be made to the various steps of the summarization process. As suggested by some users, the abstract summaries should be better organized and provide more specific information. For example, instead of generating sentences like: *The manager made negative comments about the display. She also made a decision on this matter.*, it should generate more informative sentences like: *The manager criticized the display layout and menus. They should be redesigned asap.*

Although users were in general quite satisfied with the Entity View, one possible improvement we plan to investigate is the presentation of the entities in coherent logical clusters; for instance, by following [9,20]. Another possibility to provide a quick overview of the content of the conversation could be to apply more sophisticated topic modeling techniques. However, unfortunately, the accuracy of these techniques is still unsatisfactory [8].

While our approach relies on NLP techniques that can be applied to conversations in different modalities, including emails and blogs, the visualization of the conversation itself is so far limited to synchronous conversations with a linear thread, like meetings. In the future, we shall extend the Transcript View to display non-linear asynchronous conversation threads, typical of emails and blogs. For this, we will build on previous work, such as [16,12].

References

1. Article on Tag Cloud, Wikipedia, `http://en.wikipedia.org/wiki/Tag_Cloud`
2. Bouamrane, M.-M., Luz, S.: Navigating Multimodal Meeting Recordings with the Meeting Miner. In: Larsen, H.L., Pasi, G., Ortiz-Arroyo, D., Andreasen, T., Christiansen, H. (eds.) FQAS 2006. LNCS (LNAI), vol. 4027, pp. 356–367. Springer, Heidelberg (2006)
3. Albrecht-Buehlera, C., Watson, B., Shamma, D.A.: TextPool: Visualizing Live Text Streams. In: Proceedings of the IEEE Symposium on Information Visualization, Washington, D.C., USA, pp. 215.1 (2004)
4. Carletta, J., Ashby, S., Bourban, S., Flynn, M., Guillemot, M., Hain, T., Kadlec, J., Karaiskos, V., Kraaij, W., Kronenthal, M., Lathoud, G., Lincoln, M., Lisowska, A., McCowan, I., Post, W., Reidsma, D., Wellner, P.: The AMI Meeting Corpus: A Pre-Announcement. In: Renals, S., Bengio, S. (eds.) MLMI 2005. LNCS, vol. 3869, pp. 28–39. Springer, Heidelberg (2006)
5. Criminisi, A., Shotton, J., Konukoglu, E.: Decision Forests: A Unified Framework for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Foundations and Trends in Computer Graphics and Vision 7(2-3), 81–227 (2012)

6. Dachselt, R., Frisch, M.: Mambo: A Facet-based Zoomable Music Browser. In: Proceedings of the 6th International Conference on Mobile and Ubiquitous Multimedia, Oulu, Finland, pp. 110–117 (2007)
7. Hsueh, P.-Y., Moore, J.D.: Improving Meeting Summarization by Focusing on User Needs: A Task-Oriented Evaluation. In: Proceedings of the 14th International Conference on Intelligent User Interfaces, Sanibel Island, Florida, USA, pp. 17–26 (2009)
8. Joty, S., Carenini, G., Murray, G., Ng, R.: Exploiting Conversation Structure in Unsupervised Topic Segmentation for Emails. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 388–398. MIT, Massachusetts (2010)
9. Kozareva, Z., Hovy, E.: A semi-supervised method to learn and construct taxonomies using the web. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, Cambridge, Massachusetts, pp. 1110–1118 (2010)
10. Murray, G., Carenini, G.: Interpretation and Transformation for Abstracting Conversations. In: North American ACL, Los Angeles, CA, USA (2010)
11. Murray, G., Carenini, G., Ng, R.: Generating Abstracts of Meeting Conversations: A User Study. In: Proceedings of the 6th International Natural Language Generation Conference, Trim, Co. Meath, Ireland, pp. 105–113 (2010)
12. Pascual-Cid, V., Kaltenbrunner, A.: Exploring Asynchronous Online Discussion Through Hierarchical Visualization. In: Proceedings of 13th International Conference Information Visualisation, Barcelona, Spain, pp. 191–196 (2009)
13. SIGIR 2006 Workshop on Faceted Search, `https://sites.google.com/site/facetedsearch/`
14. Stone, D., Jarrett, C., Woodroffe, M., Minocha, S.: User Interface Design and Evaluation (Interactive Technologies). Morgan Kaufmann (2005)
15. Tur, G., Stolcke, A., Voss, L., Peters, S., Hakkani-Tur, D., Dowding, J., Favre, B., Fernandez, R., Frampton, M., Frandsen, M., Frederickson, C., Graciarena, M., Kintzing, D., Leveque, K., Mason, S., Niekrasz, J., Purver, M., Riedhammer, K., Shriberg, E., Tien, J., Vergyri, D., Yang, F.: The CALO Meeting Assistant System. IEEE Transactions on Audio, Speech, and Language Processing 18(6), 1601–1611 (2010)
16. Venolia, G.D., Neustaedter, C.: Understanding sequence and reply relationships within email conversations: a mixed-model visualization. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Ft. Lauderdale, Florida, USA, pp. 361–368 (2003)
17. Viegas, F.B., Wattenberg, M., Feinberg, J.: Participatory Visualization with Wordle. IEEE Transactions on Visualization and Computer Graphics 15(6), 1137–1144 (2009)
18. Wei, F., Liu, S., Song, Y., Pan, S., Zhou, M.X., Qian, W., Shi, L., Tan, L., Zhang, Q.: TIARA: a visual exploratory text analytic system. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, pp. 153–162 (2010)
19. Wellner, P., Flynn, M., Guillemot, M.: Browsing Recorded Meetings with Ferret. In: Bengio, S., Bourlard, H. (eds.) MLMI 2004. LNCS, vol. 3361, pp. 12–21. Springer, Heidelberg (2005)
20. Yang, H., Callan, J.: Ontology generation for large email collections. In: Proceedings of the 2008 International Conference on Digital Government Research, pp. 254–261 (2008)
21. Yee, K.-P., Swearingen, K., Li, K., Hearst, M.: Faceted metadata for image search and browsing. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Ft. Lauderdale, Florida, USA, pp. 401–408 (2003)

Live Analytics Service Platform

Meichun Hsu

HP Labs
Palo Alto, California, USA
Hewlett Packard Co.
meichun.hsu@hp.com

1 Introduction

Business intelligence has traditionally focused on analysis and reporting of structured data extracted from enterprise on-line transaction processing systems. There is an increase in interest in recent years in combining traditional business intelligence with intelligence gleaned from new sources, including many new structured and unstructured data sources inside and outside of enterprises, such as social media data and signals generated by mobile devices. The advent of such big and unstructured data has ushered in a new generation of systems that couple distributed file systems with key value stores or the MapReduce processing paradigm, partly due to a common belief that SQL databases are not very effective for scaled-out unstructured data processing. This has led to a surge of interest in investing in alternative "NoSQL" data infrastructures, among them Hadoop being the most popular, potentially creating a shift in technology landscape and disrupting the dominance of the traditional SQL DBMSs. As a result, enterprises find themselves having to straddle both worlds and determine which technology is best suited for each job and moving datasets which are potentially large back and forth between these two classes of technologies. Furthermore, with additional technologies such as streaming, in-memory caching and real time analytics becoming available, there is an increasing opportunity to develop a new data analytics platform which combines and orchestrates multiple classes of technologies in solving big data analytics problem.

Our vision of the unified analytics platform is illustrated in Figure 1.[1] The platform supports multiple heterogeneous data store and query/compute platforms (structured, unstructured), integrates real time analysis over streaming data, and brings parallel computation closer to data. The platform presents a unified interface for applications to create and execute analytic data flows over a diverse collection of data engines. Internally, the platform comprises several modules including an operator library, tools to create and manipulate flow graphs, and an orchestrator to orchestrate their execution. It also includes an optimizer or a planner that transforms a logical analytics data flow to a functionally equivalent physical plan possibly spanning multiple data

[1] See [1] for a depiction of a similar vision with a stronger focus on real time business intelligence.

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 109–117, 2013.

engines taking into account many factors and objectives. The underlying data engines provide source data for the flow graphs and they are also the compute engines to execute most of the operator components in the flow graphs.

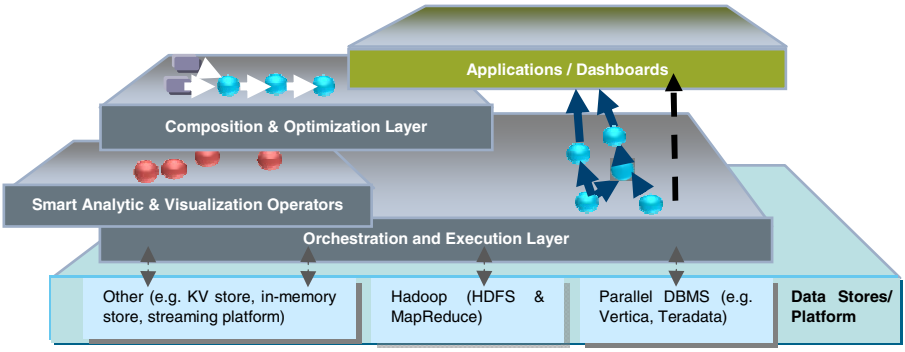


Fig. 1. Architecture and Components of Live Analytics Service Platform

This position paper provides a synopsis of some of our current work aiming at the research questions surrounding the development of such a unified platform. We focus on two classes of technologies in this paper: No-SQL, using Hadoop (the HDFS, MapReduce and Pig stack) as the specific instance, and parallel SQL databases, using the Vertica parallel DBMS as the specific instance. In section 2, we test our hypothesis that both SQL and NoSQL engines are capable of structured and unstructured data processing; we profile these alternative classes of technologies in processing unstructured data as well as combination of unstructured data and structured query processing. In section 3, we determine if there exist compelling advantages for utilizing multiple technologies for certain analytics workload, and postulate key issues on developing an optimal deployment plan for the workload. In section 4, we propose an abstraction layer which allows analytics data flows to be specified logically, which in turn allows their physical deployment, orchestration and execution on alternative technologies to be automated. We summarize the paper and outline future work in Section 5.

2 Performance Profiling of Parallel DBMS and MapReduce for Analyzing Unstructured Data

We developed a benchmark to compare and contrast performance and scalability of parallel DBMS and Hadoop/MapReduce involving processing of unstructured data. We profile alternative engines to understand how well parallel SQL Engines can process unstructured data, and how well Hadoop/MapReduce engines can process structured queries. This is particularly relevant now as several of the "analytic databases" such as Vertica, Netezza and Teradata's Asterdata are providing user defined function (UDF) frameworks that in concept allow analytics that would typically be run on Hadoop to be executed within the database. Likewise, systems such as Pig and Hive have demonstrated that relational algebra used to process

queries over structured data can be supported on top of Hadoop/MapReduce. The learnings gleaned from our benchmark framework can help further improving either technologies and help enterprises harness and optimize their investments.

We illustrate our findings with 3 tasks involving unstructured data processing as well as structured queries. Task 1 (T1) performs information extraction (IE) using the popular statistical IE method of Conditional Random Fields (CRF). The *inference* step performs the extraction task on text documents and outputs the extracted fields or "tags", and it is both data and computation intensive. The implementation on a parallel DB requires the use of a CRF user defined function. On Hadoop, CRF inference is implemented using a single MapReduce job, where the reducer performs essentially the same function as the CRF UDF. The parallel DB and Hadoop implementation take the same input file (the *document token* table) and outputs the extracted information as tags on tokens.

Task 2 (T2) uses the regular expression-based string-matching primitive; it answers the following extraction query: *find all sentences which mention a date and company "apple" together.* T2 assumes that the tokens corresponding to companies have been tagged by CRF (i.e., task T1), and it uses text string pattern matching to extract date fields. This task is a multi-step task that involves joins and filtering, but no UDF's. We express the task in Pig Latin script for testing on Hadoop, while in Vertica in a SQL statement; in either case, the regular expression matching capability is built in. T2 is further divided into 2 separate test cases: the regular-expression-matching step only, and the full multi-step workflow.

The benchmark data consists of 2.5 million sentences from about 100,000 Wikipedia articles, containing 193 million tokens and 3.9GB in size. We use a 8-node cluster that runs Red Hat Enterprise Linux, rel. 5.8, and each node has 4 Quad Core Intel Xeon Processors X5550, 48GB of RAM, and SATA disks. We evaluate the performance of the 3 tasks, along with initial loading, with 5 different scale factors scaling the size of input to up to 16x. The results are summarized in Figure 2.

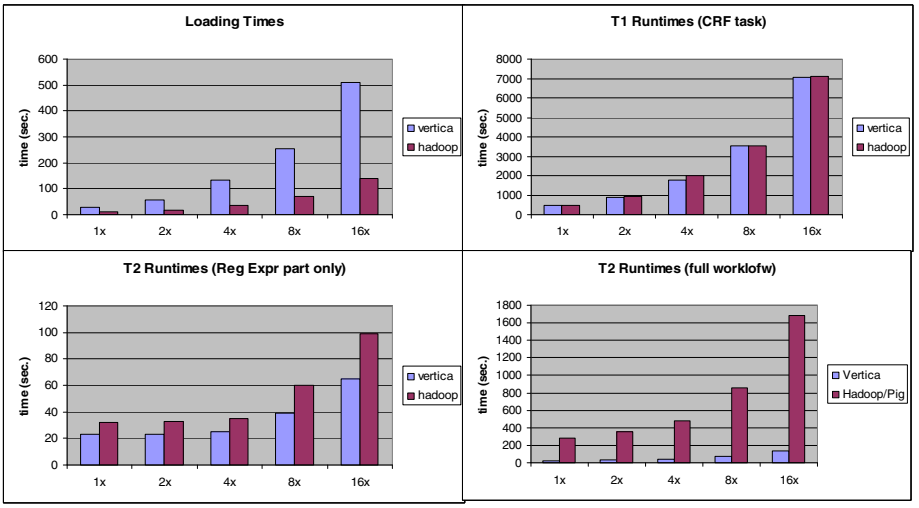


Fig. 2. (a)-(d) Performance Results of Information Extraction Processing

While many variations of configuration can be discussed (omitted here for brevity), our basic benchmark results clearly demonstrated the following[2]: (a) Complex text analytics can be as easily configured using UDFs on Vertica as on Hadoop, and they scale out well on both; (b) While data loading is faster on Hadoop File System, for more complex multi-step analytics workflows, especially those involve relational operators as steps (illustrated in our T2 with full workflow), Vertica runs far more efficiently than Hadoop/Pig; (c) It is not only feasible to perform text analytics on parallel SQL engines, these engines could be very competitive alternatives to Hadoop for unstructured text analytics; the optimal choice will depend on overall workload characteristics and cost of data movement.

3 Hybrid Deployment Strategy

Given the existence of significant performance and cost tradeoffs between the parallel DB engines and Hadoop or NoSQL engines, we set out to better understand how to leverage these two technologies and potentially consolidate more of the work into one cohesive environment. In this section we examine the possible choices in deploying an analytic workload utilizing multiple technologies, and postulate key issues on developing an optimal deployment plan for the workload. We again use the Vertica prallel DB and Hadoop/MapReduce as the two engines to be considered.

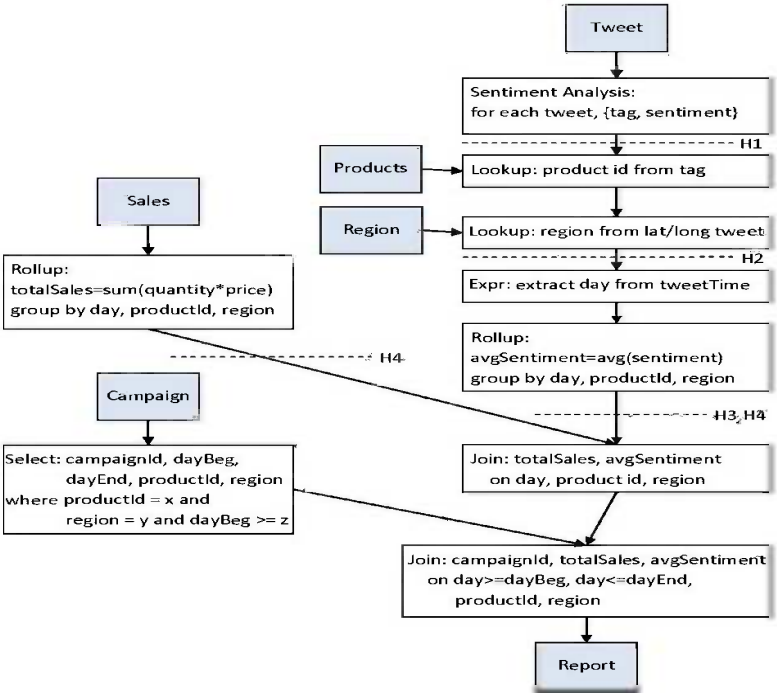


Fig. 3. Logical steps and data flow for the campaign analysis application

[2] More details of this study are documented in [2].

We develop our findings with a campaign analysis application which combines analysis of unstructured data with that of structured data. The application collects social media data (tweets) and performs sentiment analysis to determine aggregate customer sentiment by product, region and time. The result of tweet analysis is in turn joined with sales and campaign data (structured data). Figures 3 shows the analytics data flow of this application with each of the logical steps delineated.

We identify 6 possible deployment plans for this data flow. (a) *sql*: Assume all data is alerady loaded in Vertica, and execute all steps of the analytics application using Vertica alone. (b) *mr*: Assume all data resides in Hadoop file system, and execute all steps of the analytics application using Hadoop/MapReduce alone. (c) hybrid (*hb1* to *hb4*): Assume unstructured data initially resides in Hadoop file system, and structured data in Vertica; adopt a hybrid plan, where a part of the application is done on Hadoop/MapReduce, and another part is done on Vertica. Under the hybrid plan, we examine 4 logical variations, each corresponds to a "cut point" illustrated as the dotted lines in Figure 3. For example, cut point *H1* stipulates that the tweet analysis step is performed on Hadoop, and the result is transferred to Vertica to execute the remaining steps. We denote the 4 hybrid choices corresponding to the 4 cut points shown in Figure 3 (H1 to H4) as *hb1* to *hb4*.

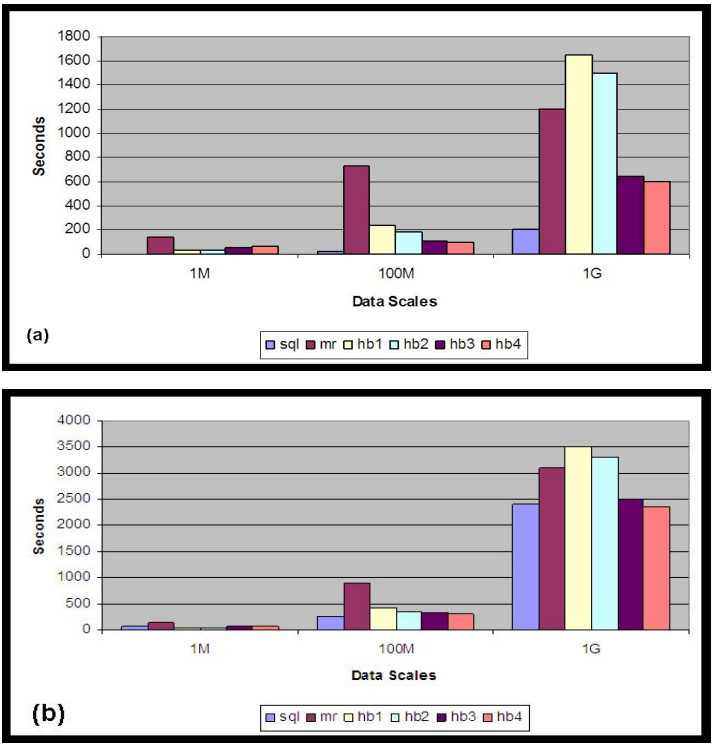


Fig. 4. Performance of single-engine plans (sql, mr) and hybrid plans (hb1-hb4)

We conducted performance experiments using 3 data sizes: 1 million tweets (1M), 100 million tweets (100M), and 1 billion tweets (1G), with the other data scaled accordingly. We use a 16-node cluster that runs Red Hat Enterprise Linux, rel. 5.8, and each node has 4 Quad Core Intel Xeon Processors X5550, 48GB of RAM, and SATA disks. We installed pDB on 8 nodes of MR on the other 8 nodes. The results for the above 6 plans are shown in Figure 4(a). We also studied the performance of these 6 plans but with the assumption that the data initially resides as files in a flat file system, and must first be loaded to the respective systems. The performance results that include initial data loading are shown in Figure 4(b).

This simple set of experiments demonstrated the following[3]: (a) Consistent with results from Section 2, if only a single engine is used, pDB is much more efficient than Hadoop/MR; however, the performance ratio narrows as the data size increases, indicating Hadoop/MR is more competitive only when data sizes are very large; (b) When data loading is not taken into account, the pDB single-engine plan outperforms any of the hybrid plans; however, among the hybrid plans, hb3 and hb4 (which does more work on mr to reduce the data size to be transferred before cutting over to pDB) perform significantly better than hb1 and hb2 (which does less work on mr to reduce the data size and thus requiring larger data transfer to cut over to pDB) as data size increases. This implies that choosing the transition point in a hybrid plan must take into account not only which engine performs the work more efficiently, but also the volume of data to be transferred. (c) When data loading is taken into account, the hybrid plans become much more competitive; this is due to the disparity of data loading overhead between the two engines. As data size increases, both data loading and data transfer are much significant factors to be considered and appropriately chosen hybrid plans will become more attractive.

4 A Service Interface for the Live Analytics Service Platform

The goal of the live analytics service platform is to offer a unified framework to harness multiple engines and data stores for analytics applications. To this end, we must enable the analytics application developers to define applications declaratively and not have the applications necessarily tied to specific engines. This will allow the application to take advantage of optimized hybrid deployment strategies and automation of execution on multiple engines. In this section we demonstrate the analytic operator and composition layers of the Live Analytics Platform shown in Figure 1, and discuss their linkage to the optimization and execution layers.

As illustrated in the example data flows in Section 2 and Section 3, the tasks or steps of the data flows, for example, the CRF function, the sentiment analysis function, or the generic select, project, and aggregate functions, may be executed on alternative engines. The platform offers a set of built-in functions as well as an ability for additional user defined functions (or operators) to be specified.

The platform supports a collection (or catalog) of analytic applications. It offers developers the ability to add operators and analytic applications; an application could be as simple as the invocation of a single operator, or be composed from multiple

[3] More details as well as many other aspects and variations of the hybrid flow study are documented in [4] and [5].

operators or nested sub-applications. An operator or an application specifies the input and output data schemas and properties. For example, the CRF operator introduced in Section 2 takes a set of tokens that belong to a sentence as input, and produces a set of tagged tokens as output. Simpler functions such as FILTER applies filtering rules tuple by tuple.

When a developer registers an operator, s/he provides an implementation of the operator. The platform provides a specification of the abstract interfaces that the implementation needs to conform to. In conforming to the interfaces, the implementation becomes deployable on the engine or engines designated by the implementation, and the execution layer of the platform can invoke the operator in the context of a data flow.

The interface to the live analytics platform's operator and application catalog is shown in Figure 5. The first few analytic applications illustrate simple operators (document converters) which take a single document as input and produce a single document as output. The *Tweet sentiment analysis service* is an application composed from two operators: *Concept extraction* and *Concept sentiment,* as shown in Figure 6.

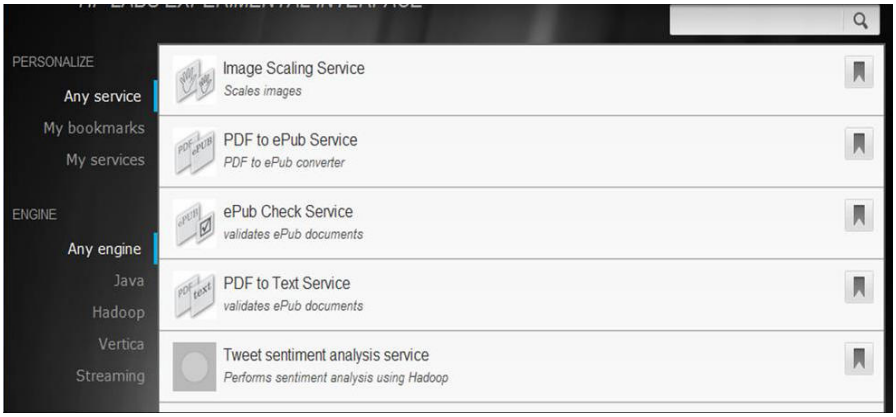


Fig. 5. Live Analytics Platform - Catalog of Operators and Applications

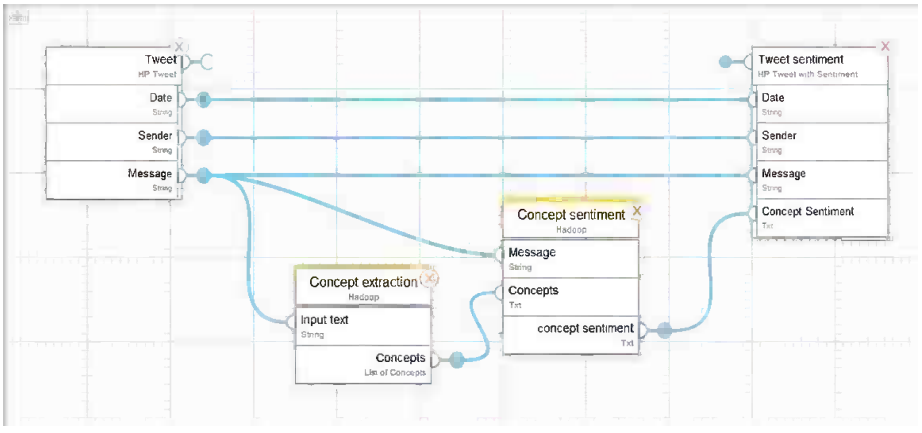


Fig. 6. An example of a composed application referencing 2 operators

A composed analytic application, such as the one illustrated in Figure 6, declaratively references built-in and user-defined functions without specifying which physical engines should be used for their execution; the platform determines the engines that are available for (or capable of) executing each of the functions, which in turn determines the possible physical deployment choices of the application.

To the extent that the functions in an analytic application are available on multiple engines, the platform chooses an execution plan based on rules and policies. As demonstrated in Section 2 and Section 3, an optimal execution plan depends on many factors including performance characteristics of the operators on each of the applicable engines, where the input and output data is located, the volume of data to be transported if multiple engines or data stores are involved, efficiency of data transportation and loading, and dynamic workload distribution on each of the engines available. It is the optimizer component of the live analytics service platform that is responsible for constructing a physical plan for the declaratively specified analytics application.

5 Summary and Future Work

In this paper, we proposed and described the live analytics service platform as an architecture for providing analytics for big data. As business intelligence evolves to big, unstructured data, we have observed the trend that open source, scale-out data analytics engines (e.g. Hadoop/NoSQL) are gaining strength among enterprises, cloud service providers, and ISVs. To understand the role of SQL and NoSQL technologies in big and unstructured data processing, in Section 2 we demonstrated that newer generation of scale-out SQL database engines are also viable choices in unstructured data analytics, while they potentially outperform NoSQL by orders of magnitude in most analytic queries that involve both structured and unstructured data. This motivated our thesis that hybrid deployment has the potential to achieve an optimal balance among multiple objectives such as cost and performance. In Section 3, we explored the characteristics of hybrid deployment strategies for analytic data flows. We demonstrated that factors such as the placement of data, the volume and latency of data movement between engines, as well as dynamic workload distributions, can significantly impact the utility of one technology vs. another in processing a complex analytic data flow. We also illustrated the operator and application framework in Section 4 of the live analytics platform, where implementations of operators can be encapsulated, while their physical deployment on different engines in the context of data flows of an declaratively specified analytics applications can be chosen by the optimizer and planner layer of the platform, and orchestrated by the execution layer of the platform.

While we have built an initial version of the live analytics platform, much work remains. In addition to continuing our work around performance benchmarking and the development of the optimizer and planner component, we also need to continue to enhance the analytics operator and application framework. In particular, while in this paper we focused on analytics/query processing over static, stored data, we are

extending the framework to streaming data (which addresses the "fast" aspect of the big, fast, total data vision). We will add streaming semantics to our operator framework, add a class of streaming engines into our platform, and study the tradeoffs and optimization strategies; an initial study on a scaled-out streaming engine is reported in [3]. We will also enhance the interface to the platform so that developers and business customers of analytic services can effectively benefit from their communities, enabling the communities to focus more on big data applications than on the complexities of the underlying technologies. A longer term direction of the live analytics platform includes introducing big data engines based on emerging technologies such as in-memory processing, and the enhancement to the abstraction layers of the platform required to take better advantage of such emerging technologies.

Acknowledgement. The Live Analytics Service Platform architecture, as well as specific work reported in this paper, was jointly developed with Malu Castellanos, Fei Chen, Qiming Chen, Umesh Dayal, Mike Lemon, Craig Sayers, Alkis Simitsis, and Kevin Wilkinson.

References

[1] Castellanos, M., Dayal, U., Hsu, M.: Live Business Intelligence for the Real-Time Enterprise. In: Sachs, K., Petrov, I., Guerrero, P. (eds.) Buchmann Festschrift. LNCS, vol. 6462, pp. 325–336. Springer, Heidelberg (2010)

[2] Chen, F., Hsu, M.: A Performance Comparison of Parallel DBMSs and MapReduce on Large-Scale Text Analytics. In: EDBT Conference (2013)

[3] Sax, M.J., Castellanos, M., Chen, Q., Hsu, M.: Performance Optimization for Distributed Intra-Node-Parallel Streaming Systems. In: ICDE SMDB (2013)

[4] Simitsis, A., Wilkinson, K., Dayal, U., Hsu, M.: HFMS: Managing the Lifecycle and Complexity of Hybrid Analytic Data Flows. In: ICDE Conference (2013)

[5] Simitsis, A., Wilkinson, K., Castellanos, M., Dayal, U.: Optimizing analytic data flows for multiple execution engines. In: ACM SIGMOD Conference, pp. 829–840 (2012)

Strategic Management for Real-Time Business Intelligence

Konstantinos Zoumpatianos, Themis Palpanas, and John Mylopoulos

Information Engineering and Computer Science Department (DISI),
University of Trento, Italy
{zoumpatianos,themis,jm}@disi.unitn.eu

Abstract. Even though much research has been devoted on real-time data warehousing, most of it ignores business concerns that underlie all uses of such data. The complete Business Intelligence (BI) problem begins with modeling and analysis of business objectives and specifications, followed by a systematic derivation of real-time BI queries on warehouse data. In this position paper, we motivate the need for the development of a complete Real Time BI stack able to continuously evaluate and reason about strategic objectives. We argue that an integrated system, able to receive formal specifications of the organization's strategic objectives and to transform them into a set of queries that are continuously evaluated against the warehouse, offers significant benefits. In this context, we propose the development of a set of real-time query answering mechanisms able to identify warehouse segments with temporal patterns of special interest, as well as novel techniques for mining warehouse regions that represent expected, or unexpected threats and opportunities. With such a vision in mind, we propose an architecture for such a framework, and discuss relevant challenges and research directions.

1 Introduction

Strategic Management (SM) is concerned with the continuous evaluation and control of a business and the environment within which it operates; it assesses internal and external factors that can influence organizational goals, and makes changes to goals and/or strategies to ensure success. SM has been practiced since the '50s, thanks to seminal contributions by Alfred Chandler, Peter Drucker and others who emphasized the importance of well-defined objectives and strategies that together determine and guide organizational activities [20]. Specific analysis techniques have been developed to support SM processes, including the strengths-weaknesses-opportunities-threats (SWOT) analysis technique widely used in practice [14].

SWOT analysis focuses on internal strengths and weaknesses, as well as external opportunities and threats that may facilitate/hinder the fulfillment of organizational objectives. Once such SWOT situations are identified, they need to be continuously monitored in real-time to assess the degree to which they

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 118–128, 2013.

occur and keep track of their evolution over time (monitoring). In addition, operational data need to be continuously scanned in search of emerging SWOT situations or unexpected data (outliers).

Being able to identify such threats and opportunities requires systems able to process data as they evolve, as well as, algorithms able to discover trends and deviations hidden among multiple layers of aggregated information. The problem of aggregating data over multiple dimensions has been studied within the domain of data warehousing, where OLAP technology [6] provides to the analysts the ability to explore and query these aggregates, in search of interesting market segments. Traditionally data warehouses considered time as a common dimension inside a data cube. Although, as Chaudhuri and Dayal noticed, a dimension of special significance [3]. Moreover, the notion of time has also been studied from the perspective of temporal databases, where it became known as temporal data warehousing [2, 16]. In such systems two distinct temporal information types are identified: the validity time and the transaction time. The validity time refers to the interval within which a record "holds", and the transaction time refers to temporal information (i.e., a point in time). For a detailed review the reader can look at [8]. Even though temporal data warehouses solved the problem of dimension and record evolution, they did not provide any means for monitoring trends and deviations in streams of data, and they also failed to capture the evolution dynamics of specific market-segments.

It becomes clear at this point that treating time as a common dimension can be rather limiting. This is especially clear in applications where predicting future incidents or the demonstration of complex behaviors over time, as well as their causes, is important. Moreover when this kind of temporal analytics have to be provided in a real-time and concise manner all over the data-set, data-warehouses have to treat the dimension of time, wherever it is found (validity time or transaction time), as a first class citizen and not solely rely on external tools for temporal data mining.

Various systems have been proposed, in order to solve this problem, that adopt time-centric data representation techniques. These include the CHAOS system, which tackles the problem taking into account the business rules and their continuous evaluation against the data warehouse [9]. Others works have concentrated on creating OLAP architectures able to handle streaming data, capture their evolution dynamics over the time [5, 10] and identify interesting segments of the data cube [15].

While all of them tackled the problem from various perspectives, to the best of our knowledge, there have been no comprehensive implementations that combine trend identification, as well as deviation detection query answering mechanisms, with the high-level information of the strategic objectives of a company. Additionally, there have been no tools that allow this kind of analytics to be done in a systematic way, allowing the automatic (and ad-hoc) generation and evaluation of such analysis queries.

The main thesis of this position paper is that taking into account SM models leads to truly effective Business Intelligence (BI), enabling the pain-free analysis

of real-time SWOT situations, a task that is critical to modern organizations. The intricacies and special characteristics of this synergy have not been carefully studied before.

In this study, we describe the development of a system that extends the traditional functionality of data warehouses in two complementary ways. First, we propose the tight integration of SM models in BI solutions. Such an integration ensures that all the important, strategic questions are evaluated, thus better aligning the needs of the business analyst with the data analytics performed on the warehouse. Second, we describe two additional classes of queries, namely, trend and deviation detection queries, which are necessary for monitoring the progress of the Strategic Objectives of an organization and perform SWOT analysis. Note that these types of queries have to be natively supported by the data warehouse (current systems do not offer this functionality), in order to allow for their efficient execution in the context of real-time BI. In addition, we discuss how these queries can be semi-automatically generated from the Strategic Objectives, reducing the burden for the analysts, and streamlining the process.

2 BI Models for Strategic Management

Consider a fictitious multinational electronic sales enterprise (hereafter ElectronicE) selling a range of products in Europe, with a strategic objective to increase its market share by 5% over the next week, as a result of a large scale search engine and social networking ad campaign (objective increase-market-share, or IncreaseMS). Because of the nature of electronic sales and electronic campaigns, the enterprise needs to be able to monitor the impact of their advertising in real-time. During strategic planning, this objective can be refined into sub-objectives through AND/OR decompositions of IncreaseMS, such as the development of new product lines (NPLines), the introduction of targeted advertizing (TAdvert), or the creation of new eCommerce channels (NChannels) for reaching ElectronicE's customer base. Out of this planning exercise, one or more strategies are adopted for achieving IncreaseMS. Each such strategy is a plan consisting of concrete actions that can be carried out by ElectronicE or its partners. SWOT analysis adds to this planning exercise a risk component. What are the factors that could affect positively/negatively the achievement of IncreaseMS? An economic downturn (EconomicD) in a region would drive sales down. Lack of knowhow (LackKH) within ElectronicE would slow down progress on the development of new eCommerce channels, while a wrong choice of target market segment (WrongMS) can affect the impact of targeted advertizing. On the other hand, a successful design for a new product (GoodD) would definitely help with the fulfillment of IncreaseMS. The elements of this scenario can be captured in a modeling language such as the Business Intelligence Model (BIM) [12]. One can think of BIM as a language in the same family as the Entity-Relationship Model, but with the concepts of 'entity' and 'relationship'

replaced by concepts such as 'goal', 'situation', 'indicator', 'process' and more. Figure 1 shows a simple BIM model for our scenario. The figure captures the concepts mentioned above, and relates them through relationships such as AND-refinement (the goal IncreaseMS is refined into three sub-goals), influence (with a + or - label to indicate whether a situation influences positively/negatively a goal) and association (each indicator is associated with at least one goal and/or situation).

Traditionally, Key Performance Indicators are used for monitoring strategies and measuring their success or failure. They are simplistic measures that we would like to include in our model for identifying the way they can be combined for modelling complex goals and situations. Specifically, we use such indicators in our modelling process for providing input from the data warehouse to our complex strategic models. Formally, Indicators (short for Key Performance Indicators, or KPIs) measure the degree of fulfillment of a goal, or the degree of occurrence of a situation, and possibly other things (e.g., the quality of a product or the degree of completion of a process). For example, the increase in sales over a period of time (SalesC) measures how well we are doing with respect to the objective IncreaseMS, while a negative change to Gross Domestic Product (GDP) measures the degree to which we have an economic downturn situation in a geographic region. In a similar vein, lack of knowhow is measured by delays in developing new eCommerce channels (indicator time-to-completion, or TimeTC). Indicator values are often normalized so that they fall in the range [-1.0, 1.0]. Moreover, thresholds are provided to mark whether a goal is fulfilled, partially fulfilled, denied, etc. [1].

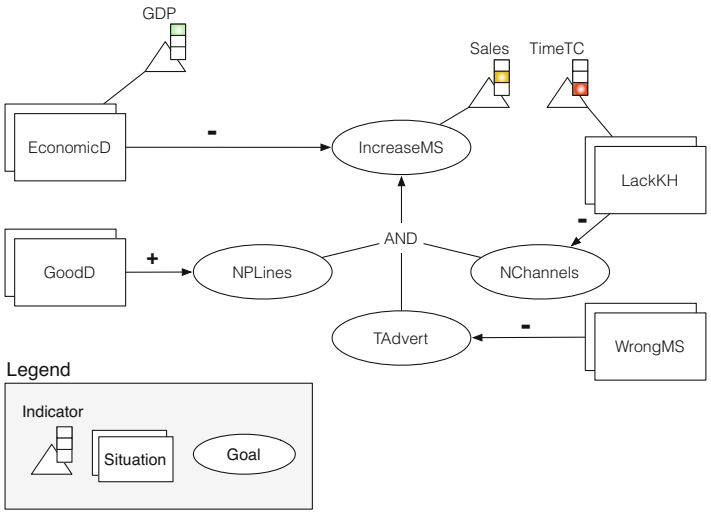


Fig. 1. A simplified strategic model

3 Analysis for SM

There are two facets to the analysis we want to support. First, there are dimensions to the problem at hand (fulfilling IncreasedMS) that identify with the dimensions of the underlying data cube, in our case (i) time (over what periods did/didn't we do well?), (ii) region (Trentino/Italy/Europe) and product (specific product/product line/all products). This means that for any question we want to answer, such as "How are we doing with IncreaseMS?" there will have to be series of queries that answer the query at different levels of granularity. Such example queries could be the following: *"What are the market segments that are affected by our strategy?", "Is the trend in the affected market segments expected or unexpected?", "Why are we doing bad or well within some specific market segments?".*

Speficically, the analysis to be conducted can have three different modalities:

1. **Monitoring and Impact** – this modality applies when we are trying to keep track of the status of an objective or situation, i.e., we are trying to answer "how are we doing?" questions: *Are we on track with IncreaseMS? Has there been any change to the status of situation EconomicD? What is the impact of a locally applied strategy?*
2. **Outliers** – here we are looking for surprises, both positive and negative. Such a modality is very important for organizations, since it ensures that interesting information that is not part of the strategic model arrives to the analyst. For example: *"Sales have jumped in Lombardia threefold relative to the rest of Italy".*
3. **Explanation and Troubleshooting** – in this modality, we are analyzing the reasons for some observed behaviors. For example, if we are doing badly or very well with respect to IncreaseMS (Monitoring and Impact modality), we need to pinpoint where (what regions? ... time periods? ... for what products?) and for what reasons this happens. The reasons would be the root causes for observed performance, such as weaknesses and threats that correlate with observed performance. Additionally, the identification of outliers needs to be followed by diagnosis (why?) and possible followup action, remedial or perfective (replicate in other regions a successful strategy).

It is important to emphasize the importance of the outliers modality of analysis. Peter Drucker and others have pointed out since the 70s that we live in an Age of Discontinuity where extrapolating from the past is hopelessly ineffective [7]. Accordingly, strategic analysis should not fall into the linearity trap and always look out for surprises, because they are now the norm.

4 SM Enabled Business Intelligence

Monitoring a large, high-dimensional data cube for patterns of interest, as well as for unexpected patterns, is not a trivial task. It is hard to analyze such

data cubes at the most detailed level, because of their sheer size, and at the same time summarization techniques, which can reduce the amount of data to operate on, may hide various interesting patterns (e.g., anomalies) that could be identified as threats or opportunities. In systems where we are interested in supporting continuous trend queries (e.g., checking for the fulfillment of the Strategic Objectives of a company), it is of great importance to be able to capture the fine-grained temporal dynamics of each market segment within the data cube, so that we are able to perform queries considering the temporal aspects of our data.

Moreover, current data warehouses rely on pre-specified hierarchies for each dimension in the database, for example, the City, Region, Country hierarchy. Nevertheless, within the current global market such hierarchies may not always make sense to consider and, additionally, their full extent may not be known beforehand. This means that new interesting groupings could be discovered that would complement the analysis based on the existing dimension hierarchies.

Building a system able to cope with a high workload of continuous monitoring queries corresponding to SWOT analysis, leads to the following set of challenges.

1. Real-time updating of the data warehouse does not necessarily mean instantaneous updates [27]. Instead, the update process should follow the business requirements. This means that we need to be able to assess how often we need to update which data in regards to our goals and their current status. This is something that we can judge from our Strategic Model and the evolution dynamics of the data currently available in the data warehouse.
2. The system should include mechanisms for generating a set of focused exploratory queries based on the Strategic Model, as well as, algorithms for answering them efficiently. Such queries should allow for the **Modeling and Impact** Strategic Management analysis modality. Their systematic evaluation should be decided based on the Strategic Model and with efficiency in mind (i.e., decide *which* queries to execute *when*). Further on we should be able to support the **Explanation and Troubleshooting** analysis modality by providing mechanisms that are able to explain why the status of our goals and indicators is positive or negative with regards to our expectations. This means that we need to be able to provide insight for the correct exploration of the data cube behind the data that affect the specific parts of the Strategic Model.
3. Monitoring the Strategic Objectives of an organization is vital, but it is tricky since there may exist Threats and Opportunities that do not directly affect the Strategic Model. For this reason, multidimensional outlier detection techniques have to be employed, such that we identify all the interesting parts in the data: these techniques should use as little input information as possible, and (as pointed out earlier) go beyond the dimension hierarchies explicitly defined in the data warehouse. In order to fully support both the **Outliers** and the **Explanation and Troubleshooting** analysis modalities, explanation techniques have to be deployed in order to explain such deviations in the data.

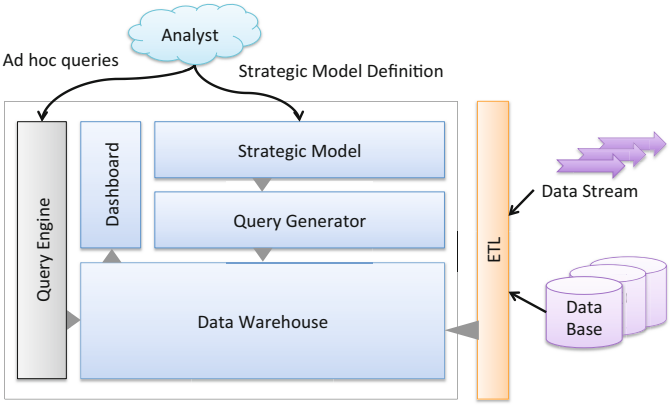


Fig. 2. System architecture

Our solution proposal, depicted in Figure 2, involves a Strategic Model layer that the user defines based on the business schema, a Query Generator layer that creates continuous queries in regards to the model, as well as a Query Engine that is able to answer trend and outlier detection queries on top of a specialized and efficient Data Warehouse structure. The architecture also includes a Dashboard for the visualization of results (which can also be generated from the Strategic Model [21]). We elaborate on these components in the following paragraphs.

The Strategic Model layer allows the user to specify the Strategic Objectives of the organization, the indicators that affect them, as well as their interconnections and decompositions. Additionally, the ability to perform ad-hoc queries to the system is given to the analyst, so that queries can also be formed manually. The Query Generator layer is responsible for automatically generating continuous queries based on the Strategic Model. These queries are evaluated against the data warehouse, in order to provide feedback with respect to the Strategic Goals in real time. Recent research [25] has shown that it is possible to generate data warehouse queries from the Strategic Objectives of an organization for monitoring its Strategic Goals. Such techniques have to be extended and incorporated in this layer, in order to interpret high level formalizations into low level operations, such as trend and outlier detection queries, on top of the data warehouse.

The Data Warehouse layer is responsible for storing the data received as soon as they are available (requirement 1), ideally in a streaming fashion. This means that the warehouse has to be designed in such a way that it can keep expanding as data arrive, with the minimum storage and processing cost overhead. Moreover since we are interested in performing fine-grained temporal queries on top of our data, it has to be able to inherently support trend and outlier detection queries using a temporal representation of the data available in each cell. At this point we argue in favor of the use of time series based representations for each market segment, in order to efficiently capture and reason about temporal dynamics.

Finally, the Query Engine has to be able to answer the queries generated by the Query Generator, as well as the ones that where directly submitted by the user. This component incorporates specialized time-series based algorithms for identifying trends, aggregating parts of the cube, as well as identifying outliers within the data cubes. An example trend query is the following: "Will the current sales trend that we observe up to now, within a time window W, in the market segment S help as achieve the goal of increasing our market share by 5% (IncreaseMS)?". Additionally, an outlier detection query could be of the form "Are there any interesting sub-markets within Europe?". The results of such a query could be unexpected patterns, such as sub-markets in Europe where the company is doing surprisingly worse than others, or expected patterns such as a sub-market that is responding to the company's strategy.

It is also important that the Query Engine is able to compute and monitor trends, and identify outliers with respect to *ad hoc* market segments, not relying on pre-specified hierarchies and groups, as they need manual setup and may not always be representative of the data. An example could be the Countries, Cities hierarchy, where we cannot always expect that all sales within a country or a city follow the same patterns. As a result, we should be prepared to find outliers in groupings not explicitly specified by the hierarchy, for example it may be the case that interesting trends appear in regions within a country.

5 Previous Work

Data warehouses support decision making by providing On-Line Analytical Processing tools (OLAP) [6] for the interactive analysis of multidimensional data [11]. Such tools allow a person (analyst) to quickly acquire important information drilling in and out of the most interesting aggregates of a database, a task that is fairly complex considering the large number and sizes of dimensions [24]. The common case for updates to a data warehouse is the Extract-Transform-Load (ETL) processing [26], i.e., data are extracted from the sources and loaded to the data warehouse during specified time intervals. As applications are pushing for higher levels of freshness, data warehouses are updated as frequently as possible, giving rise to Active Data Warehousing [13, 23, 22].

While traditional data cube technology is good for static aggregates, it fails to explain trends in the multi-dimensional space [5] (requirement 2). For this reason, linear regression analysis in time series based data cubes has been presented in [10, 5]. The basic idea was to do linear regression analysis in the maximum level of detail an analyst is willing to inspect, in order to identify the trend of the data in each cell and use these low level trends for calculating the ones in the upper layers. Furthermore, the efficient generation of logistic regression data cubes was studied in [28], where the authors introduced an asymptotically lossless compression representation (ALCR) technique, as well as an aggregation scheme on top of it, in order to eliminate the need of accessing the raw data for the calculation of each distinct cell in the data cube.

With regards to the problem of reducing the size of a time series based OLAP data cube, a tilted time-frame scheme aiming at reducing the storage costs, as

well as a popular-path based partial materialization of the multi-dimensional data cube for reducing the processing time are presented in [5]. Additionally in [9] an extended Haar wavelet decomposition technique has been proposed for reducing the amount of data, as well as, for providing a reduced hierarchical based synopsis method for the time domain.

There has been very little work on identifying the interesting parts of a time series based data cube. In [5] the idea of exceptional regression lines has been presented, i.e. regression lines with a slope greater than or equal to an exception threshold. Moreover, in [15] the problem of identifying the top-k most anomalous parts of a regression-lines-based cube has been studied. The main observation, on which their solution was based, was that subjects which are parts of a hierarchy, should behave similarly to the parent market segments. Even though this method will work for the hierarchies existing in the warehouse, we envision an approach able to identify a wider range of anomalies (e.g., anomalies not directly correlated to the existing hierarchies).

We note that combining the use of temporal data representations as first class citizens, techniques for outlying market segments identification, and the Strategic Objectives modeling, gives rise to a complex problem that none of the current works addresses. CHAOS [9] is the system that is the most relevant to our proposal. This system is able to create a multi-dimensional cube for streaming data, apply event processing above it, and visualize the interesting parts of the cube. It is able to answer queries of the form: *"Report the set of biggest changes within a market segment M and a time window W"*. Using the results of such queries it is able to evaluate complex business rules and infer events of special significance. Nevertheless, the business rules that CHAOS considers are defined relative to the data warehouse schema, and not relative to the business schema, creating a disconnect to the Strategic Objectives of the organization. Moreover, CHAOS does not describe any techniques for trend and outlier analytics, which are indispensable for performing SWOT analysis and answering SM queries.

Additionally a set of related works that could be used on top of our proposed system are the ones related to Sentinels [19] and the OODA concept [17]. The OODA concept describes the loop of Observation (are the data normal?), Orientation (what is wrong?), Decision (user analysis) and Action (course of action) on top of a set of KPIs. Sentinels are causal relationships between KPIs mined in the data warehouse which are used to trigger early warnings, thus helping the analyst perform OODA cycles efficiently. Such sentinels could be mined and integrated in our Strategic Model as situations that can affect our goals. Related to that is the work on Bellwether analysis [4] which aims at identifying parts of the data that form good predictors. This means parts of the data that have low overhead and high prediction accuracy. Bellwether analysis can be used on top of our framework for picking good predictor features for identifying the future values of our Indicators. In regards to the problem of allowing a user to modify the view of a system and define his own strategies and queries, there has been recent work on what is known as meta-morphing [18].

6 Conclusions

In this work, we motivated the need for a system that is able to continuously monitor a data warehouse based on queries generated from the Strategic Model of an organization. This system should be able to identify trends in regards to these pre-specified objectives, and also to monitor the warehouse for expected or unexpected threats and opportunities in the data as well as their causes. Furthermore, we presented the related work and its gaps, we discussed the challenges of building such a system and presented an architecture of a system that treats the trends and outlier detection techniques, in all the layers of the system, as first class citizens, which is crucial for the kind of analysis we propose.

Acknowledgements. This research was partially funded by the FP7 EU ERC Advanced Investigator project Lucretius (grant agreement no. 267856).

References

[1] Barone, D., Jiang, L., Amyot, D., Mylopoulos, J.: Composite indicators for business intelligence. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 448–458. Springer, Heidelberg (2011)
[2] Chamoni, P., Stock, S.: Temporal structures in data warehousing. In: Mohania, M., Tjoa, A.M. (eds.) DaWaK 1999. LNCS, vol. 1676, pp. 353–358. Springer, Heidelberg (1999)
[3] Chaudhuri, S., Dayal, U.: An overview of data warehousing and OLAP technology. ACM SIGMOD Record 26(1) (March 1997)
[4] Chen, B., Ramakrishnan, R., Shavlik, J.W., Tamma, P.: Bellwether analysis: Searching for cost-effective query-defined predictors in large databases. ACM TKDD 3(1) (March 2009)
[5] Chen, Y., Dong, G., Han, J., Wah, B.W., Wang, J.: Multi-dimensional regression analysis of time-series data streams. In: VLDB, vol. 02 (2002)
[6] Codd, E.F., Codd, S.B., Salley, C.T.: Providing OLAP (on-line Analytical Processing) to User-analysts: An IT Mandate, vol. 32. Codd & Date, Inc. (1993)
[7] Drucker, P.F.: The age of discontinuity: Guidelines to our changing society. Harper and Row, New York (1968)
[8] Golfarelli, M.: A survey on temporal data warehousing. International Journal of Data Warehousing 5 (2009)
[9] Gupta, C., Wang, S., Ari, I., Hao, M.: Chaos: A data stream analysis architecture for enterprise applications. In: CEC (2009)
[10] Han, J., Chen, Y., Dong, G., Pei, J., Wah, B.W., Wang, J., Cai, Y.D.: Stream Cube: An Architecture for Multi-Dimensional Analysis of Data Streams. Distributed and Parallel Databases 18(2) (2005)
[11] Jarke, M., Lenzerini, M., Vassiliou, Y., Vassiliadis, P.: Fundamentals of Data Warehouses. Springer (2003)
[12] Jiang, L., Barone, D., Amyot, D., Mylopoulos, J.: Strategic models for business intelligence. In: Jeusfeld, M., Delcambre, L., Ling, T.-W. (eds.) ER 2011. LNCS, vol. 6998, pp. 429–439. Springer, Heidelberg (2011)
[13] Karakasidis, A., Vassiliadis, P., Pitoura, E.: ETL queues for active data warehousing. In: IQIS 2005 (2005)

[14] Lamb, R.: Competitive strategic management. Prentice-Hall, Englewood Cliffs (1984)
[15] Li, X., Han, J.: Mining approximate top-k subspace anomalies in multi-dimensional time-series data. In: VLDB (2007)
[16] Mendelzon, A.O., Vaisman, A.A.: Temporal Queries in OLAP. In: Proceedings of the 26th International Conference on Very Large Databases (2000)
[17] Middelfart, M.: Improving business intelligence speed and quality through the ooda concept. In: DOLAP 2007 (2007)
[18] Middelfart, M., Bach Pedersen, T.: The meta-morphing model used in TARGIT BI suite. In: De Troyer, O., Bauzer Medeiros, C., Billen, R., Hallot, P., Simitsis, A., Van Mingroot, H. (eds.) ER Workshops 2011. LNCS, vol. 6999, pp. 364–370. Springer, Heidelberg (2011)
[19] Middelfart, M., Pedersen, T.B.: Implementing sentinels in the targit bi suite. In: ICDE (2011)
[20] Nag, R., Hambrick, D.C.: What is strategic management, really? Inductive derivation of a consensus definition of the field. Strategic Management, 955 (2007)
[21] Palpanas, T., Chowdhary, P., Mihaila, G., Pinel, F.: Integrated model-driven dashboard development. ISF 9(2-3) (July 2007)
[22] Palpanas, T., Sidle, R., Cochrane, R., Pirahesh, H.: Incremental maintenance for non-distributive aggregate functions. In: VLDB (2002)
[23] Polyzotis, N., Skiadopoulos, S., Vassiliadis, P., Simitsis, A., Frantzell, N.E.: Supporting Streaming Updates in an Active Data Warehouse. In: ICDE (2007)
[24] Sarawagi, S., Agrawal, R., Megiddo, N.: Discovery-Driven Exploration of OLAP Data Cubes. In: Schek, H.-J., Saltor, F., Ramos, I., Alonso, G. (eds.) EDBT 1998. LNCS, vol. 1377, pp. 168–182. Springer, Heidelberg (1998)
[25] Souza, V.E.S., Garrigós, I., Trujillo, J.: Monitoring Strategic Goals in Data Warehouses with Awareness Requirements. In: ACM Symposium on Applied Computing (2012)
[26] Vassiliadis, P., Simitsis, A., Georgantas, P.: A generic and customizable framework for the design of ETL scenarios. Information Systems (2005)
[27] Watson, H.J., Wixom, B.H., Hoffer, J.A., Anderson-Lehman, R., Reynolds, A.M.: Real-time business intelligence: Best practices at continental airlines. Information Systems Management 23(1) (2006)
[28] Xi, R., Lin, N., Chen, Y.: Compression and aggregation for logistic regression analysis in data cubes. In: TKDE (2009)

Pricing Approaches for Data Markets

Alexander Muschalle[1], Florian Stahl[2], Alexander Löser[1], and Gottfried Vossen[2]

[1] TU Berlin, FG DIMA, Einsteinufer 17, 10587 Berlin, Germany
forename.surname@tu-berlin.de
[2] University of Münster, ERCIS, Leonardo-Campus 3,48149 Münster, Germany
forename.surname@ercis.de

Abstract. Currently, multiple data vendors utilize the cloud-computing paradigm for trading raw data, associated analytical services, and analytic results as a commodity good. We observe that these vendors often move the functionality of data warehouses to cloud-based platforms. On such platforms, vendors provide services for integrating and analyzing data from public and commercial data sources. We present insights from interviews with seven established vendors about their key challenges with regard to pricing strategies in different market situations and derive associated research problems for the business intelligence community.

1 Introduction

The analysis of freely available data, together with commercial and in-house data, is an increasing market segment. One example is market research focused analytics of Web data, with the aid of natural language processing technologies and statistical methods. In order to analyze data sets of such size large IT infrastructures need to be built. However, this is potentially costly as such systems usually have high implementation costs, as well as significant further costs for updating and analyzing data. Even though, we can reduce the problem to a few hundred GB that will fit into main memory, these companies still need capable staff in their IT department who can maintain and program complex in-memory multi-core infrastructures [22]. In particular, for small and medium enterprises (SME) the associated risks with such infrastructures are a strong barrier for innovation.

Arguably, cloud computing could lower this barrier as far as operating the hardware is concerned. It has to be taken into account that, though cloud computing infrastructure might be operated at lower cost, it is still likely to be to expensive for a single SME to rent the hardware necessary to crawl and analyze significant portions of the Web. This is why SMEs are not yet in a position to benefit from the latest research in cloud computing and Web mining. Nevertheless, SMEs often hold unique assets for transferring data into business relevant information, e.g., domain knowledge of a particular niche or relationships to potential customers that are willing to pay for the information.

Only recently, vendors of data, providers of data warehouse solutions, and algorithm providers have started to offer their products as platform-, software-,

M. Castellanos, U.Dayal, and E.A. Rundensteiner (Eds.): BIRTE 2012, LNBIP 154, pp. 129–144, 2013.

and data-as-a-service on so called data market places. These data markets supply analysts, business applications, and developers with meaningful data and an eased data access. Moreover, for data producers these marketplaces act as single integration platform. As a result, data marketplaces enable completely new business models with information and analysis tools as electronically tradable goods. The collective storage, analysis, and utilization of data from the Web on a central platform offers many cost savings and high innovation capabilities. Thus, especially for SMEs significant market entry barriers and impediments to innovation are eliminated.

In this paper we present an empirical study with seven data market owners and producers of data associated products and services. In this preliminary study with early adaptors, we collected answers from our interview partners for the following important questions:

1. What are common queries and demands of participants on a data market?
2. Which pricing models utilize beneficiaries for data associated products?
3. Which research challenges for the business intelligence community may arise from the combination of data and data associated services in data markets?

The reminder of this paper is structured as follows: In Section 2 we report on two query categories and seven types of beneficiaries that are common across all interview partners. In Section 3 we discuss the market situation of our interview partners, reveal current pricing strategies and derive major research challenges in Section 4. In Section 5 we discuss related work and conclude in Section 6. Finally in Appendix A we give a brief introduction to our research methodology.

2 What Is a Data Market?

In this section we review common elements of business models and business demands of our partners. We start with queries that a data market should be able to answer and then report our observations about seven beneficiaries and their demands.

2.1 Common Query Demands

During our interviews we asked our interview partners for common demands of their customers. We observed a heterogeneous set of queries from which we could derive the following two major scenarios:

Estimate the Value of a 'Thing', Compare the Value of 'Things'. In the first scenario, customers abstract the data market as a data warehouse that resides on an open set of data sources. These customers utilize the data market for collecting and measuring factual signals from public data sources, such as the Web or 'Open Data' from the UNO, and non-public sources, such as commercial data from Reuters and in-house private data, such as data from an ERP or CRM system. Given the set of available data sources on the data market and

the data warehouse-like processing infrastructure, a common user demand is formulating key indicators which describe the value of an immaterial or material good. Finally, the user utilizes these key indicators, together with data sources and the storage and integration platform of the data warehouse, for computing a value of immaterial and material goods and for ranking these goods by this value function. Among many others, our interview partners mentioned the following interesting scenarios:

Ranking Influencers on the Social Web. *Which top-10 Web forums influence buying decisions of German BMW motorbike customers? What are the most influential users?* [1]

Value of a Web Page for Placing Advertisements. *On which highly ranked and frequently visited Web pages for 'men's underwear' should I buy advertisement space?* [2]

Value of a Starlet or Impact of a Politician. *Order politicians of the German parliament by the number of mentions in major German newspapers. Does ex-tennis star 'Steffi Graf' reach the right customer set for an advertisement campaign for fast food products?*

Value of a Product. *What are top-10 journals for 'my' life science publications? A banker will issue question like: Is the product of a loan requesting company really accepted by their customers? What opinions do their customers state about the product?*

In these scenarios 'fact tables' contain measurable key indicators from publicly available Web data, while dimension tables may contain data from commercial vendors or in house private data, such as product lists. Note, that in this scenario the 'schema' of these dimension tables is formalized through the schema of these commercial or in house data sources. Moreover, all scenarios share the need for aggregating key indicators from the Web and for correlating these key indicators with a monetary transaction, such as decisions on costly marketing measures. Moreover, our interview partners agreed, that their users only require a partial order of the value of objects. Therefore, users are — to a certain extend — willing to tolerate uncertain data samples, such as sentiment and factual data extracted from textual sources.

Show All about a 'Thing'. The second class of scenarios is about collecting factual information about a certain thing and integrating this factual information into a universal relation. For decades, this scenario is well known in the information integration community, see [6], but also in the text mining community, see [11] and information retrieval community, see recent work on exploratory search [18], [17]. Our interview partners consider the ability of the data warehouse owner to resolve and reconcile logical objects across a set of heterogenous sources as a major research challenge.

[1] A prominent data collector for answering such queries is the page 'Klout.com'.
[2] See also alexa.com and other vendors of products for search engine optimization.

2.2 Beneficiaries and Participants

Throughout our interviews we could identify seven types of beneficiaries that may issue queries against and may profit from the services of a data market. Figure 1 shows the relationship between beneficiaries and services.

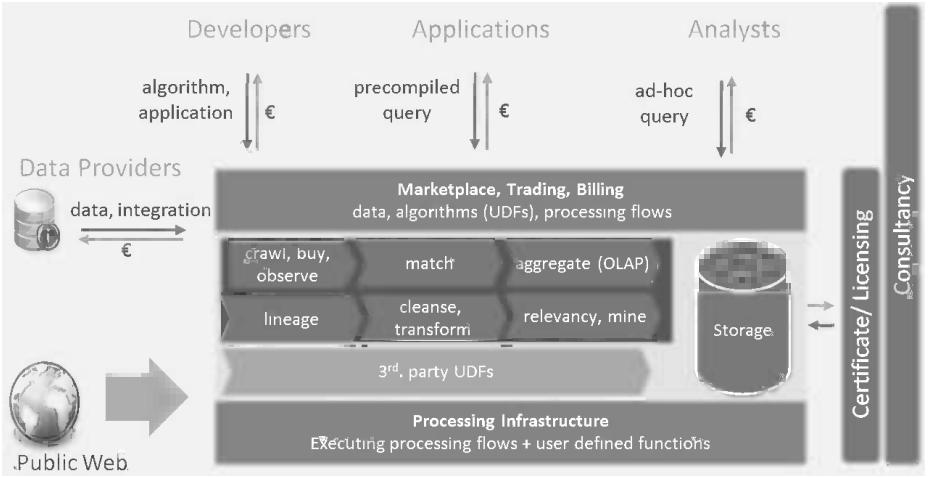


Fig. 1. This figure shows a general schema of a data marketplace for integrating public Web data with other data sources. In analogy to a data warehouse architecture, the schema includes components for data extraction, data transformation and data loading, as well as meta data repositories describing data and algorithms. In additional, the data marketplace offers interfaces for 'uploading' and methods for optimizing, potentially complementary, black box operators with user-defined-functionality, as well as components for trading and billing these black box operators. In return, the 'vendor' of the user-defined-function retrieves a monetary consumption (indicated by the euro symbol) from buyers. Moreover, in the case of large data volumes from the Web, the marketplace relies on a scalable infrastructure for processing and indexing data.

Understanding demands, interests, and needs of different beneficiaries is crucial before we can analyze potential technological challenges. Only then can we start building systems that may solve their problems. In a further step we can think about pricing strategies and prices that these participants are willing to pay. From the interviews we learned about the following seven groups:

1. **Analysts.** Typical members of this group are domain experts, such as M&A experts, sales executives, product managers, brokers, marketing managers and business analysts. The most often used data exploration tools for these experts are Web search engines. To a lesser extend this group utilizes 'traditional' OLAP focused business intelligence tools. Moreover, this group utilizes office products, mostly for aggregating and summarizing results from a data exploration into a meaningful report. From a data market perspective, this group tries to benefit from the sheer endless options of combining

publicly available (Web) data, commercial data (commercial sources) and private (enterprise) data. To do so, this group issues ad-hoc queries against data sources and combines data in a highly interactive way, thereby formalizing knowledge about data sources and data integration steps through their interactions with a data market place. This group is of high relevance for other participants on the data market, since analysts create a demand for frequently requested data and associated data related services through ad-hoc queries.

2. **Application vendors.** Often analyst may not be comfortable in expressing their demands in a formalized machine readable representation. Additionally, the task of data exploration requires many repetitive and labor intensive steps. Application vendors formalize common requirements from analysts into applications. These applications ease data access for a broader group of domain experts. Important examples are business analytics applications [12], mash-up applications [23], customer relationship management applications, and enterprise resource planning applications. These applications formalize common data processing knowledge into a sequence of pre-compiled queries, such as queries for selecting sources or queries for integrating and aggregating data. From a pricing perspective of a data market, pre-compiled query sequences have a high potential to create stable, continuous demands for data and services.
3. **Developers of data associated algorithms.** Both previous beneficiary types, application vendors and analysts, need to integrate data, often from a large and volatile set of sources. Examples are algorithms for data mining, matching, cleansing, relevance computation and lineage tracing. These algorithms vary for different data domains, text data languages, data 'quality' or data 'joinability', and often only fit specific niche domains. Developers may upload these algorithms to a data marketplace as a black box user-defined-function, so other participants of the data marketplace may 'buy' and try out these algorithms. Algorithm developers may buy 'shelf' space and may rent the infrastructure from the owner of the data marketplace.
4. **Data providers.** We learned that our interview partners distinguish between commercial and non-commercial providers of Web data, for example Web search engine owners, such as Bing or Google, Web archive owners, providers of linked data from the Web and Web forum owners who seek commercialization of their forum content. The interviewees also mention government agencies, like the UNO or the World Bank, which provide statistics free of charge. Also, they mention commercial data providers, such as Reuters or Bloomberg among others, which have a long history in the area of selling financial data and geo-data. Data providers utilize market places for storing and advertising data. Some data providers also started offering data integration algorithms.
5. **Consultants.** Our interviewees frequently mentioned consultancy services for supporting analysts in tasks like data source selection, integration, evaluating and product development.

6. **Licensing and certification entities.** On a data market this group assigns a 'branded label' to data, applications and algorithms that may help customers when buying data related products.
7. **Data market owner.** The owner of a data market place is faced with many technical, ethical, legal and economic challenges. Probably the most important economic challenge is establishing a trusted brand and large community. An important technical challenge is developing a common platform for storing, searching and exchanging data and related algorithms, while minimizing algorithm execution time. In particular in Europe, legal aspects may represent significant market entry barriers for trading data across European countries.

In the next section we discuss the different market situations for these beneficiaries. Later (in Section 4), we derive interesting and novel research challenges for the data warehouse and business intelligence community from these market situations.

3 Market Situations and Pricing Strategies

In our interviews we observed that prices for data and associated services mainly depend on demanders' preferences and market structures. This section describes common market structures and associated pricing strategies we observed from our interview partners.

3.1 Current Market Structures

Relevant in the context of this paper are three market structures — *monopoly, oligopoly,* and *strong competition.* In a **monopoly** a supplier holds enough market power to set prices to a level which maximizes profits. In order to do so, suppliers sell fewer quantities at higher price. A monopolist is well advised not to set a single price for a product, but different prices for different demanders (as different demanders have different preferences). This is commonly referred to as price discrimination.

However, these preferences are mostly hidden so that the monopolist tries to motivate demanders to reveal their preferences. This could be done by pricing in respect of quantity, entry level degree, social grouping, etc. In particular in a data market environment the monopolist approach is to set a price, learn from the demanders' reaction by slightly increase/decrease the prices. This enables a monopolist to shape the demand function based on which the supplier is able to maximize profits, by perfect price discrimination.

When monopolists loose their position because one or more competitors enter the market, an **oligopoly** is created, i.e., the market is dominated by a few. This can change the former monopolist's situation dramatically. Effects range from price fights to pooling of interests. Substantial knowledge of the specific industry and complex game theory analysis can potentially help forecasting players' behavior.

Under **strong competition** market prices tend to align with marginal costs. This trend is enhanced especially when marketplaces come up because they naturally promote transparency by enabling demanders to compare products. Thus, individual suppliers lack the market power of setting a profit-maximizing price, but have to face the market price (i.e., selling either for the market given price or do not sell at all). The low marginal costs of data and algorithms can lead to problematic price developments because the overall costs are not relevant for short term decisions. As long as suppliers realize positive gross margins, it is rational to do the transaction at prices near marginal costs. In the long run, this approach is dangerous, because overall costs have also to be covered. As long as suppliers are not able to affect the demanders in a way, that they perceive a unique product quality, suppliers will have to decrease prices. Tweaking overall costs will be the only measure to avoid profit loss. This progress is nothing inherent only to markets of Data as a Service, but are typical in any competitive market with low marginal cost.

We discovered — even among our limited number of interviewees — that the perceived competition differs significantly. Some interviewees were able to mention their *strongest competitors* instantaneously and can thus be categorized as being in a strong competitive environment. Others felt there is no directly competing product or service (*monopolist*). The remaining interview partners knew about similar offerings to their products but were not particular concerned about them. Overall, our interviewees concordantly consider competition as an adequate means to forward welfare and market size growth. Our interviewees consider the market (to date) as big enough and players do not put too much pressure onto each other. Especially in one-to-one competition we found out that there is no willingness to start a price fight.

3.2 Observed Pricing Strategies

It is a common defection, especially in a merely technical domain, to think that prices should be based on costs. In fact, all of our interviewees consider costs only as a limiting factor of reasonable pricing. Rather, they agree that it is almost only the demanders' preferences that determine prices. To what extent suppliers are able to influence a demander's preferences is an issue of marketing operations. A deliberate pricing strategy often turns out to be a major contribution to gain profits rather than any cost reducing measure. According to the interviewees' statements we observed four main categories of pricing models:

1. **Free** data can be obtained from public authorities, such as statistical data[3]. Our interviewees argue that free data can unlikely be sold for money only because it is offered on a market place. Nevertheless, free data available on one's marketplace can help to attract customers which in turn attracts suppliers of commercial data. Moreover, free data can be integrated with in-house or private data and this integration could become valuable.

[3] `http://data.gov.uk/`

2. **Usage based prices** correspond to the human rationality that each single unit of a commodity raises the total amount of money to pay for. We observed that our interview partners utilize usage based prices for consultancy time or API calls; as an example, in our interview series several partners charged consultancy services per hour. This volume based approach substantially loses power of persuasion if marginal costs converge to zero. Our interview partners expect a trend of falling marginal costs in their product portfolio and expect expect that volume based pricing on markets for data and algorithm may no longer become the first choice among marketplace owners.
3. **Package pricing** refers to a pricing model that offers a customer a certain amount of data or API calls for a fixed fee. For one of our interview partners offering API-based marketing research this was the model of choice. Other vendors, such as Yahoo!BOSS[4] or OpenCalais[5], a Reuters subsidiary, also offer this pricing scheme. Interestingly, none of our interviewees uses APIs together with pure usage based pricing. One reasons could be that these companies sell quantities that are not actually used by the customers or that that accounting overhead is too high. Note that, depending on the package size, API-calls potentially allow the model of arbitrage. Optimizing package based models is a subject of current research, see [14,15].
4. The **flat fee tariff** is one of the simplest pricing models with minimal transaction costs. It is based on *time* as the only parameter. Among our interview partners, we observed this pricing scheme mainly in regard to software licenses and software hosting. On the one hand, a flat fee tariff provides suppliers and demanders with more safety in planning future activities. This rests upon the fact that time is linear. On the other hand, especially for the demander's side, a flat fee tariff lacks flexibility. A supplier carefully has to bear in mind his product's specific market structure and the demander's preferences. To do so, suppliers could combine a flat fee tariff with flexibility by offering short term contracts.
5. A combination of the previous two is **two-part tariff** pricing. In this scenario customers pay a fixed basic fee and on top of that an additional 'fee per unit consumed'. This pricing scheme is also commonly used by telephone companies. In the data scenario we figured out that one interview partner utilizes the fixed part to cover the fixed costs, where as the variable fee generates the profit. Another example are pricing schemes for software license where prices are calculated by taking a base fee and adding an surcharge depending on the numbers of users who would use the system.
6. **Freemium** is another approach of pricing data and algorithm on marketplaces. The idea is to let users join and use basic services for free and charge them for (premium) services that provide additional value to customers. The payment model for additional services can take any of the forms described above. A couple of our interview partners take exactly that line to ensure a

[4] `http://developer.yahoo.com/search/boss/`
[5] `http://www.opencalais.com/`

large attendance to their services. One interview partner realized that there is no value in simply reselling data but rather in offering additional services for data integration . We bear in mind, though, that such a price strategy only works if product's marginal costs are very small, otherwise per unit losings could get out of control.

3.3 The Effect of Product Substitutions on Pricing Strategies

The marginal rate of substitution is an important parameter for pricing strategies in many different market structures. The marginal rate of substitution indicates what quantity of good A is equivalent to a quantity of good B. A perfect substitute (marginal rat of substitution of 1), implies that demanders are completely indifferent wether to buy product A or product B. Under real conditions, perfect substitutes are rare. However, suppliers on marketplaces for data and algorithm have to understand in how far demanders consider the supplier's products or services as substitutable. For example, if a supplier trades a specific tagging algorithm as a monopolist, another supplier with a completely different algorithm, but identical benefit from demander's point of view (tagged text), could gain the whole market, by setting a price slightly below the monopolist's one.

4 Challenges for the Business Intelligence Community

Our interviews resulted in five trends in the area of data markets for the near future. In this section we present for each trend attractive research opportunities for the business intelligence community.

4.1 Growing Number of Data Providers

The first trend is a *growing number of data providers*. One example are Web forum and Web page owners that seek commercializing user-generated-content. Currently, these parties usually sell advertisement space. In the future, these parties will also sell raw and homogenized content or even enrich and aggregate content, thereby allowing for analytical insights. Another example are governmental offices, such as `data.gov.uk` or `www-genesis.destatis.de`. In the past, these organizations mainly offered publicly available reports, often as printouts. Only recently these organizations have started to offer online analytics. Finally, publicly funded initiatives, such as multimillion projects triggered by the European Union or financed by venture capital, will collect and aggregate data and will seek options for commercializing data. Examples are a `www.mia-marktplatz.de`, a market for data of the German Web or `datamarket.com`, a company that mainly collects, integrates, aggregates, and visualizes public available data from governmental institutions.

Challenge 1: Create information extraction systems that can attach structured, semantically meaningful labels to text data with little human effort. Given that labeled text data, enable an analyst OLAP operations on Web data with the same simplicity as a Web search.

4.2 Data Markets Will Offer the Entire Stack for Niche Domains

Recent technological advances, such as sophisticated implementations (e.g., PACT/Nephele [3] implementation of the map/reduce principle [10]), distributed storage and querying architectures (such as HBASE or SenseiDB), or high level batch processing languages (like Pig-Latin[20] or JAQL [4]) drastically simplify the access to even large commodity clusters of data storage and processing nodes. The availability of this technology in the open source community 'potentially' enables each marketplace owner to host the growing number of available data sources. Therefore, *competition and diversification among data markets* will raise, which our interview partners consider as the second important trend. They argue that data markets will provide not only data, but will soon start to offer data associated algorithms and data visualizations. Our interviewees consider to reach different beneficiaries groups as another option for diversification. One example are data market places, such as `infochimps.com` or `factual.com`, which mainly focus on the group of data developers and which provide application programmer interfaces for this group. Our interviewees argue that these market places will soon also serve analysts. Developing such a domain specific stack for niche domains is difficult and requires *tools for creating mash-ups and data supply chains* (like DAMIA[23], KAPOW, or Yahoo!Pipes).

Challenge 2: Enable analysts to create domain specific data processing flows with little costs. Enable and optimize black-box user defined functions in these flows.

4.3 Customers Demand Data More Quickly

From our interviews and market observation we notice a trend towards more brand monitoring. That means that the appearance of brands of consumer goods and also of institutions such as universities are regularly monitored on the Web. This is done towards different ends. On goal is to analyze how a brand is perceived by customers. Another goal is to react to negative comments in order to reduce the harm. In particular the last example falls into the category of publish-subscribe patterns. Companies are only realizing what is possible with Web monitoring and are likely to demand the services in near realtime in the near future. This development is analogous to realtime BI but more complex as the amount of data is significant larger.

Challenge 3: Build systems that can reliably answer brand monitoring queries to indexes, with heavy read and write access, sticking to ACID constraints on a scalable infrastructure in order to enable near realtime brand monitoring.

4.4 Customers Will Use Product Substitutes

Data markets do not recognize that some products can be substituted by others. That is particulary true for data associated algorithms, such as text mining libraries for extracting factual and sentiment information from user-generated

content. As a result, it remains difficult for application developers to identify appropriate data algorithms, in particular for niche data sets. Worse, analysts and developers cannot determine how a particular algorithm is different from competing products. Ideally, customers of data and associated algorithms could try out algorithms on a selected data set before buying. A standardization of data processing mash-ups (see also Challenge 2) would enable product substitution, which in turn would enable competition between suppliers. Generally, competition was seen as very positive by all of our interviewees.

Challenge 4 (example substitution): Given a list of entities and their properties, identify a set of mining algorithms that have been optimized to a very high degree for exactly this data set. Provide a user with an execution sample and recommend an algorithm with regard to execution time, precision and recall.

4.5 Data Markets Will Offer Price Transparency

On the consumer side the main motivating factor for using data market places is to be able to buy products or services at the right price and in the right quality from a single source. Providing price transparency to users would force data suppliers and algorithm developers to optimize their data sets and algorithms. This may lead to an increase in customers on the market place which again is the most attractive factor for suppliers. Therefore, price transparency can lead to a positive development of the entire market place usage and may increase the revenue for the data market operator. However, for an individual supplier this transparency would eventually cause a drop on sales, since customers of this supplier will eventually substitute the more expensive product with a lower priced product of another supplier.

Challenge 5: Develop incentives for suppliers for price transparency.

4.6 Learn from Your Customers

On a data market place, information about data, algorithms and customer behavior is available on a single platform. This information enables a data marketplace to track and derive customer preferences, which are valuable for suppliers. The market place operator could sell this 'secondary' data which would provide another revenue stream. On the economic side, making this information available will bring a data market place closer to a perfect market (i.e., a market on which all information is transparent and available to everyone; one market close to this is the stock market). Moving closer to a perfect market will optimize processes and prices and thus optimize the overall profits and welfare. More than anything else, a broad user base can attract suppliers who offer their data and algorithms on a data marketplace.

Challenge 6: Collect transactional data from customers. Leverage this data for solving Challenges 1-5.

Further Challenges. In this paper we focus on pricing strategies. However, we do recognize that that there are more challenges to be addressed, most notably issues of trust and privacy in a cloud-based environment.

5 Related Work

In this section we review publications on pricing of information and data goods in the field of databases as well as in the economic literature.

5.1 Pricing Information Goods

In [2] research areas related to data markets are outlined and a research agenda for the database community is derived. According to the authors, the emergence of data markets bears two challenges regarding the pricing of information or data goods: One major challenge is understanding how the value of data is modified through actors on data markets and which pricing models and services facilitate these data markets. The second challenge is understanding the behavior of market participants and the rules underlying data deals. The authors attribute the first area to the database community and describe the second area rather to the economics community. We, however, believe that this disregards an important basic economic fact: Prices, as soon as a market place is used, converge to a given market price, leaving little scope for price variations.

Most work in this area bases on the assumption that providers are able to set prices in a monopolistic way. Three main strategies exists: flat fee pricing, per use pricing and two-part tariff pricing. Regarding profit maximization the authors of [25] point out that under certain assumptions (monopoly, zero marginal and monitoring costs, homogeneous customers) flat fee pricing and two-part tariff pricing are on par, while *two part tariff* is the most profitable pricing strategy, when consumption is heterogeneous. In [5] it is demonstrated that contingency prices are best if the value of an information good is underestimated, resulting of the uncertainty of demanders as they are not able to experience the quality of the information before buying it.

5.2 Arbitrage, Pricing per Tuple, and Cost Optimizations

Considering Arbitrage. The authors of [2] identify two data markets pricing schemes, (a) a subscription based pricing with a query limit and (b) a schema where the price is determined per data set ('tuple'). The same authors discuss four major shortcomings of current pricing schemes: (1) Current price models allow for arbitrage, (2) models base on the assumption that all data sets are of equal value, (3) data acquired once has to be either cached by the customer or paid for again and (4) data providers receive no guidance on how to set their prices. We argue that the first three weaknesses of pricing schemes only occur because of the last shortcoming and propose in [15] an algorithm which validates that a pricing model is free of arbitrage Later, the same authors of [2] also suggest

modeling prices in a fine-grained way, i.e., attached to a small data unit (cell, tuple, etc.), and rules about how to transform these prices in course of a query, as this allows for maximum flexibility. They suggest using data provenance to achieve this goal. At the same time, the authors realize that computing theses prices is potentially complex. We argue that this approach does not take into account the fact that data of an arbitrary cell on its own may often have little to no value. If the attached price is equal to the cost of producing the data, one could use the suggested model at least to derive the cost of data and use this as lower bound in price negotiations.

Charging Users for Optimizations across Queries. The authors of [14] investigate how a system could charge users for optimizations across queries. Later, the authors of [24] used a game theory approach where users actively bid for an optimization while the system is value-maximizing and cost-recovering. Such an optimization is enacted if the overall utility (sum of bids minus cost) is maximal. Users are then charged according to their bid. The approach excludes users without a bit from the optimization to give an incentive to reveal true preferences. The authors of [13] suggest a cost estimation model which allows for adaptive shifting from crawling to search and vice versa. While this model estimates the costs of the service, it does not consider the value of the data for customers. In the context of cloud computing the authors of [21] introduce a method to dynamically generate prices upon request. They suggest to use these prices in negotiations or auctions: If a buyer requests data, the seller calculates a price sends it back and buyers may then choose whether to accept or not.

6 Conclusion

The increasing interest in data markets to lever all kinds of available data, public as well as private, in order to create novel consumer and enterprise value is clearly visible. This preliminary study shows seven beneficiaries of data and data associated services. For each of these beneficiaries we discussed potential market situations, pricing approaches and trends. Major research challenges are developing an infrastructure and tools that enable entire enterprises and even individuals to identify insight effectively, from their collection of data assets and from data collections on cloud-based data market places.

Although this study already shows valuable insights, it is limited by the number of participants. In our future work we will continue our study with a broader sample. The disruptions discussed in this study present exciting opportunities for the business intelligence community. We started to address hard problems, the solution of which can greatly impact the future course of data platforms and tools such as identifying text mining services for niche data [7], or investigating data processing infrastructures for large scale data such as [1].

Acknowledgements. We like to thank the following partners in our interviews: Yasan Budak, CFO and owner, Vico Research & Consulting GmbH; Stefan Geissler,CEO and owner, TEMIS Deutschland GmbH; Holger Duewiger, CTO,

Neofonie GmbH; Hjalmar Gislarson, CEO and owner, datamarket.com; Ian Mulvany, VP Product Management, mendeley.com; Christoph Tempich, Manager, inovex GmbH and Phillip Mueller, VP Strategy at Thomson Reuters. Alexander Löser receives funding from the German Federal Ministry of Economics and Technology (BMWi) under grant agreement "01MD11014A, 'A cloud-based Marketplace for Information and Analytics on the German Web (MIA)"

References

1. Alexandrov, A., Battré, D., Ewen, S., Heimel, M., Hueske, F., Kao, O., Markl, V., Nijkamp, E., Warneke, D.: Massively parallel data analysis with pacts on nephele. PVLDB 3(2), 1625–1628 (2010)
2. Balazinska, M., Howe, B., Suciu, D.: Data markets in the cloud: An opportunity for the database community. PVLDB 4(12), 1482–1485 (2011)
3. Battré, D., Ewen, S., Hueske, F., Kao, O., Markl, V., Warneke, D.: Nephele/pacts: a programming model and execution framework for web-scale analytical processing. In: SoCC, pp. 119–130 (2010)
4. Beyer, K.S., Ercegovac, V., Gemulla, R., Balmin, A., Eltabakh, M.Y., Kanne, C.-C., Özcan, F., Shekita, E.J.: Jaql: A scripting language for large scale semistructured data analysis. PVLDB 4(12), 1272–1283 (2011)
5. Bhargava, H.K., Sundaresan, S.: Contingency pricing for information goods and services under industrywide performance standard. J. Manage. Inf. Syst. 20(2), 113–136 (2003)
6. Bleiholder, J., Naumann, F.: Data fusion. ACM Comput. Surv. 41(1) (2008)
7. Boden, C., Löser, A., Nagel, C., Pieper, S.: Factcrawl: A fact retrieval framework for full-text indices. In: WebDB (2011)
8. Bryman, A., Bell, E.: Business Research Methods. Oxford University Press (2007)
9. Corbin, J.M., Strauss, A.L.: Basics of qualitative research: techniques and procedures for developing grounded theory. Sage Publications, Inc. (2008)
10. Dean, J., Ghemawat, S.: Mapreduce: simplified data processing on large clusters. In: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation, OSDI 2004, vol. 6, p. 10. USENIX Association, Berkeley (2004)
11. Doan, A., Ramakrishnan, R., Vaithyanathan, S.: Managing information extraction: state of the art and research directions. In: SIGMOD Conference, pp. 799–800 (2006)
12. Galhardas, H., Florescu, D., Shasha, D., Simon, E., Saita, C.-A.: Declarative data cleaning: Language, model, and algorithms. In: VLDB, pp. 371–380 (2001)
13. Ipeirotis, P.G., Agichtein, E., Jain, P., Gravano, L.: To search or to crawl?: towards a query optimizer for text-centric tasks. In: SIGMOD Conference, pp. 265–276 (2006)
14. Kantere, V., Dash, D., Gratsias, G., Ailamaki, A.: Predicting cost amortization for query services. In: SIGMOD Conference, pp. 325–336 (2011)
15. Kushal, A., Moorthy, S., Kumar, V.: Pricing for data markets
16. Kvale, S., Brinkmann, S.: InterViews: Learning the Craft of Qualitative Research Interviewing. Sage Publications (2008)
17. Löser, A., Arnold, S., Fiehn, T.: The goolap fact retrieval framework. In: Aufaure, M.-A., Zimányi, E. (eds.) eBISS 2011. LNBIP, vol. 96, pp. 84–97. Springer, Heidelberg (2012)

18. Marchionini, G.: Exploratory search: from finding to understanding. Commun. ACM 49(4), 41–46 (2006)
19. Myers, M.D.: Qualitative Research in Business & Management. Sage (2008)
20. Olston, C., Reed, B., Srivastava, U., Kumar, R., Tomkins, A.: Pig latin: a not-so-foreign language for data processing. In: SIGMOD Conference, pp. 1099–1110 (2008)
21. Püschel, T., Neumann, D.: Management of cloud infastructures: Policy-based revenue optimization. In: Nunamaker Jr., J.F., Currie, W.L. (eds.) ICIS, p. 178. Association for Information Systems (2009)
22. Rowstron, A., Narayanan, D., Donnelly, A., O'Shea, G., Douglas, A.: Nobody ever got fired for using hadoop on a cluster. In: HotCDP 2012 - 1st International Workshop on Hot Topics in Cloud Data Processing (2012)
23. Simmen, D.E., Altinel, M., Markl, V., Padmanabhan, S., Singh, A.: Damia: data mashups for intranet applications. In: SIGMOD Conference (2008)
24. Upadhyaya, P., Balazinska, M., Suciu, D.: How to price shared optimizations in the cloud. Proc. VLDB Endow. 5(6), 562–573 (2012)
25. Wu, S.Y., Banker, R.D.: Best pricing strategy for information services. J. AIS 11(6) (2010)

A Appendix: Methodology of Qualitative Interviews

We reported on a study of qualitative interviews with executives of seven European companies that are active in the data market area. In this section we describe our interview methodology.

A.1 Introduction to Interview Techniques

The methodology of this study implements a seven stages approach to interview research as suggested in [16]. The first step is *thematizing the study* followed by *designing the study in consideration of the knowledge to be achieved*. The subsequent steps are *interviewing*, *transcribing*, *analyzing*, *verifying*, and the last step is *reporting/publishing*. Generally it can be distinguished between interviews in quantitative and qualitative research. Whereas quantitative interviews are usually fully structured, qualitative interviews are unstructured or semi-structured [8]. In structured interviews the interviewer has pre-set an interview schedule with questions that are kept simple to allow for easy coding and comparison of results. Thereby, interviewees serve as information delivering subjects. Contrary, qualitative research interviews regard interviewees as participants [16]. Qualitative interviews can be distinguished in almost unstructured and semi-structured interviews. While the first can be regarded as type of conversation, the second follows a guide of predefined open questions or topics. Some authors even encourage to depart from the interview guide and discuss tangents to get as much insights as possible [8].

A.2 Sampling and Interview Process

We selected seven interview partners in executive positions who are involved with developing, consulting, or running data related products and service. In the

course of the interviews we extended the list of potential interviews by means of snowballing (i.e., the recommendation of potential new interview partners by interviewees [8]). Our interview partners cover a broad spectrum of the data related products and services, like social media monitoring, text enrichment, consulting or data market places. Often, an interviewee covered more than one of the given areas.

We opted for semi-structured interviews [16] to ensure that all participants underwent a similar interview and to allow for a minimum of comparability. We conducted the interviews via telephone and visualized questions via slides. The main purpose of the slides was ensuring that interviewer and interviewee had a common point of reference. Our interviews covered three topics: First, we asked the interview partners for their current position and experience. Next, we issued questions about their products and business models. Finally, we asked direct questions to examine what an optimal data market place should look like in order to fulfill the needs for the interview partner. The interviews took – on average – 63 minutes.

A.3 Analysis and Verification

In order to preserve the interview we recorded and transcribed the interviews in full [16][8]. For the actual analysis we used a grounded theory approach [9,19,8]: First, we identified codes (i.e., themes such as *data has no intrinsic value*); then summarized them in categories (e.g., *factors influencing the price*), next examined relationships between categories and finally drew conclusions from that (e.g., *a market place should offer transparent pricing shemes*). For achieving validity by reaching consent in a conversation or discourse [16] we gave the study results to the interviewees for their approval and seek scientific discourse by publishing this work.

Author Index