On-Demand ELT Architecture for Right-Time BI: Extending the Vision

Florian Waas
EMC, USA (Florian.Waas@emc.com)

Robert Wrembel
Poznań University of Technology, Poland (Robert.Wrembel@cs.put.poznan.pl)

Tobias Freudenreich
Technische Universität Darmstadt, Germany(freudenreich@dvs.tu-darmstadt.de)

Maik Thiele
Technische Universität Dresden, Germany (maik.thiele@tu-dresden.de)

Christian Koncilia
University of Klagenfurt, Austria (koncilia@isys.uni-klu.ac.at)

Pedro Furtado
University of Coimbra, Portugal (pnf@dei.uc.pt)

ABSTRACT

In a typical BI infrastructure, data, extracted from operational data sources, is transformed and cleansed, and subsequently loaded into a data warehouse where it can be queried for reporting purposes. ETL — the process of extraction, transformation, and loading, is a periodic process that may involve an elaborate and rather established software ecosystem. Typically, the actual ETL process is executed on a nightly basis, i.e., a full day's worth of data is processed and loaded during off-hours. Depending on the resources available and the nature of the data and the reporting, ETL may also be performed more frequently, e.g., on an hourly basis.

It is desirable to reduce this delay further and ideally provide reports and business insights at real-time or near real-time. However, this requires overcoming throughput bottlenecks and improving latency throughout the ETL process. Instead of attempting to incrementally improve this situation, we propose a radically different approach: leveraging a data warehouse's capability to directly import raw, unprocessed records, we defer the transformation and cleaning of data until needed by pending reports. At that time, the database’s own processing mechanisms can be deployed to process the data on-demand.

This technique ensures that resources are utilized optimally rather than spent speculatively on processing, potentially irrelevant, data in bulk. Besides excluding irrelevant data from processing all together, a more likely scenario is the case where different types of reports are run at different times of the day, week, or month and require different categories of source data. Again, using an on-demand approach helps optimize resource utilization and improves data freshness substantially.

In addition to running periodic reports, modern BI architectures also have to incorporate events in order to detect outliers or encounter exceptional situations during data processing. To capture events, we augment our on-demand approach with an active component that performs lightweight data screening independent of the ETL processing and may be integrated with a BI dashboard.

Besides outlining an overall architecture, we also developed a roadmap for implementing a complete prototype using conventional database technology in the form of hierarchical materialized views.

Keywords: data warehouse, business intelligence, data stream analysis, ETL, ELT, on-demand ETL, near-real time BI, right-time BI.

INTRODUCTION

Business Intelligence (BI) has long been considered an integral part of any successful enterprise's data processing and analysis strategy (Chaudhuri et al., 2011). BI analysts inspect and query the data, made available through a *Data Warehouse* to gain insight into sales data or other business facts that will aid them at making business decisions.

Data warehouses are periodically populated or refreshed with data from *Operational Data Stores* (ODS), e.g., front-end transaction databases. In most businesses, the freshness of the information available in the data warehouse translates directly into more timely business decisions and competitive advantage. Therefore, it is highly desirable to have data available for analysis at real-time or *near* real-time, i.e., provide data, so no delay is discernable. The degree of delay acceptable depends on the specific application scenario and actual real-time processing in the sense of sub-second delays is generally not needed. This subjective timeliness requirement is sometimes referred to as *right-time BI* (Davis, 2006).

The biggest hurdle to satisfying right-time BI latency requirements is the data processing needed to make data available in a data warehouse: the data coming from the ODS infrastructure needs to be processed before it is suitable for BI for a variety of reasons. For example, a data warehouse typically consolidates a multitude of different ODS with different schemas and metadata, hence, all incoming data must be normalized. Also, the ODS may contain erroneous or corrupted data that needs to be cleaned and reconciled. This preprocessing is commonly known as *Extract-Transform-Load* (ETL): data are first *extracted* from the original data source, then *transformed* including normalization and cleansing and finally *loaded* into the data warehouse. For simplicity, we refer to the entire ETL process as *loading* in the following, unless indicated otherwise. Figure 1 depicts a typical architecture, including various data sources, an ETL layer, and components of the reporting pipeline.

While database technology for data warehousing has seen tremendous performance and scalability enhancements over the past decade in the form of massively parallel database architectures, ETL has improved in scalability and performance to a much lesser degree. As a result, most BI infrastructures are increasingly experiencing an ingest bottleneck: data cannot be furnished to the data warehouse at the necessary pace and freshness. Clearly, in order to provide near real-time or right-time BI this bottleneck needs to be resolved.

A natural approach would be to scale the different components involved in ETL individually. In particular, parallelizing the transformation phase is instrumental in achieving better overall throughput. However, a parallel ETL infrastructure turns out to be a double-edged sword: while the processing time of daily loads may be reduced, the cost of the initial investment and, more

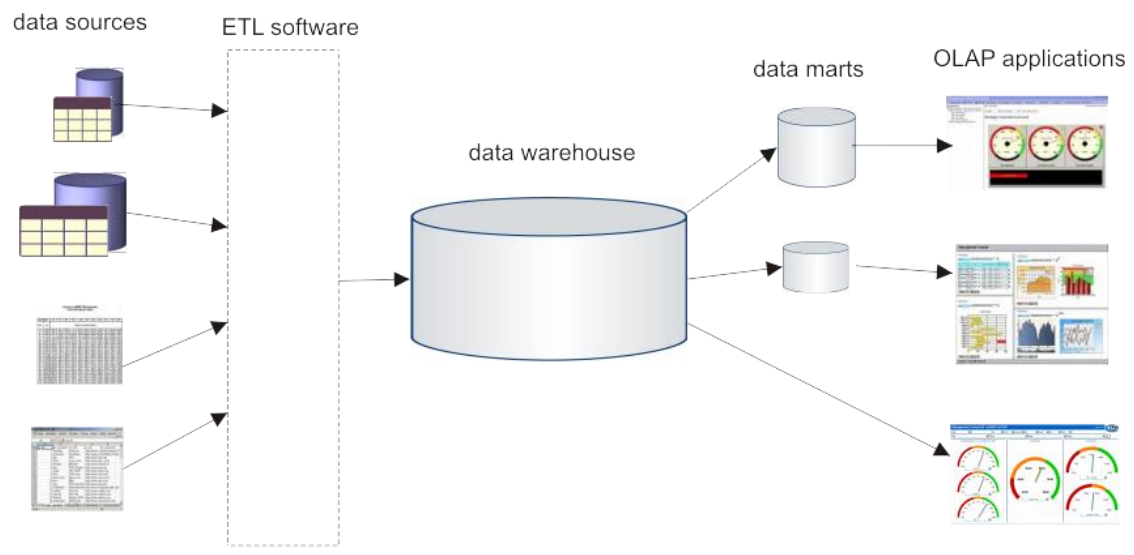

Figure 1: Typical DW architecture

importantly, the continual maintenance of a complex parallel system quickly outweigh its benefits.

Instead we propose the three following major building blocks to address real-time/right-time data acquisition.

1. **Turning ETL into ELT**, i.e., loading raw data directly into the data warehouse and then transforming it using database queries unifies the processing platforms and avoids having to build out additional complexity on the front-end. Rather, the production system is put to good use and also serves as ETL processor. This simplifies the furnishing of data and seamlessly taps into the power and capabilities of the data warehouse infrastructure. In ELT, the entire transformation process is expressed in SQL, which has significant advantages. DBAs and data engineers are intimately familiar not only with the programming language but can also tap into resources already available for trouble-shooting and performance tuning for the regular data warehouse operations.
2. **On-demand processing.** Different BI processes, i.e., reports or analytic processing, require different data sets at different points in time over an extended period. Herein lays a significant opportunity to reduce the latency of the process and simultaneously improve resource utilization. Instead of trying to condense a highly resource intensive process as a whole, we can accomplish better response time and resource utilization by selectively identifying the data needed to answer currently posed queries and processing these data sets in an on-demand fashion. This results in a model that effectively leverages pipelining instead of bulk processing.
3. In addition, we propose to supplement this architecture with a **monitoring component** that screens incoming data and alerts components down-stream according to a predetermined set of rules. This helps to ensure that consumers can be aware of data, even if queries demanding the data have not been executed yet. Finally, we identified seven main open research problems in the area of real-time/active/right-time data warehousing.

The remainder of this paper is organized as follows. First, we present standard architecture models for BI and Event Processing and survey related work. Second, we present our ELT-based on-demand architecture and present possible implementations thereof. Next, we present further directions for future work and open research problems. Finally, we conclude the paper.

BI INFRASTRUCTURES

A traditional BI infrastructure is typically composed of five layers, as shown in Figure 1. The first one — an operational data sources (ODSs) layer represents heterogeneous and distributed data sources. ODSs can include various relational databases as well as non-database storage systems. The second layer, an ETL layer, is responsible for loading data into a central database, called a data warehouse (DW) (Kimball & Caserta, 2004). Data loading is organized in several steps that include: (1) reading and filtering data from ODSs, (2) transforming data into a common data model that is used at a DW, (3) cleansing data in order to remove inconsistencies, duplicates, and null values, (4) integrating cleansed data into one consistent set, (5) computing summaries, and (6) loading data into a DW. ETL tasks are executed periodically (e.g., nightly, during off-hours) and may involve an elaborate and rather established software ecosystem. The third layer includes an already mentioned data warehouse that stores integrated and summarized data. The fourth layer is composed of repositories (called data marts), typically storing subsets of aggregate data from the central DW. Each data mart includes data specific to a given business domain, e.g., marketing, finance, and HR (Golfarelli & Rizzi, 2009). The fifth layer — an on-line analytical processing (OLAP) layer, is responsible for various types of data analysis and visualizations (e.g., reporting, dashboards, management cockpits). OLAP applications typically execute complex queries to discover trends, patterns of behavior, and anomalies as well as for finding hidden dependencies between data.

Due to the high performance and scalability capabilities of modern DWs, BI system designers can profit from an ELT architecture. By contrast to ETL, ELT typically uses standard data transfer mechanisms, e.g., FTP, to transfer data directly into a DW, where the raw data are stored in a temporary storage, commonly referred to as *staging area*. The integration, transformation, and data cleansing tasks are then applied on the data by means of a data warehouse management system's (DWMS) internal mechanisms, including SQL commands and stored procedures. Such processed data are then loaded into proper DW tables. Figure 2 presents a BI architecture based on ELT. Notice that the data warehouse stores both raw data from ODSs (possibly with some intermediate temporary data) as well as cleansed and integrated DW data.

The advantages of ELT compared to ETL include among others: (1) easier management of data provenance and drill-through implementation as all the raw and a DW data are stored in the same database, (2) greater throughput gained by means of a DWMS parallelization and tuning, (3) increased data processing efficiency by means of advanced query optimization offered by a DWMS.

Nonetheless, even with the ELT architecture the delay between data updates at ODSs and seeing the effects of the updates by the applications may be too high for some applications (e.g., the detection of unauthorized credit card usage).

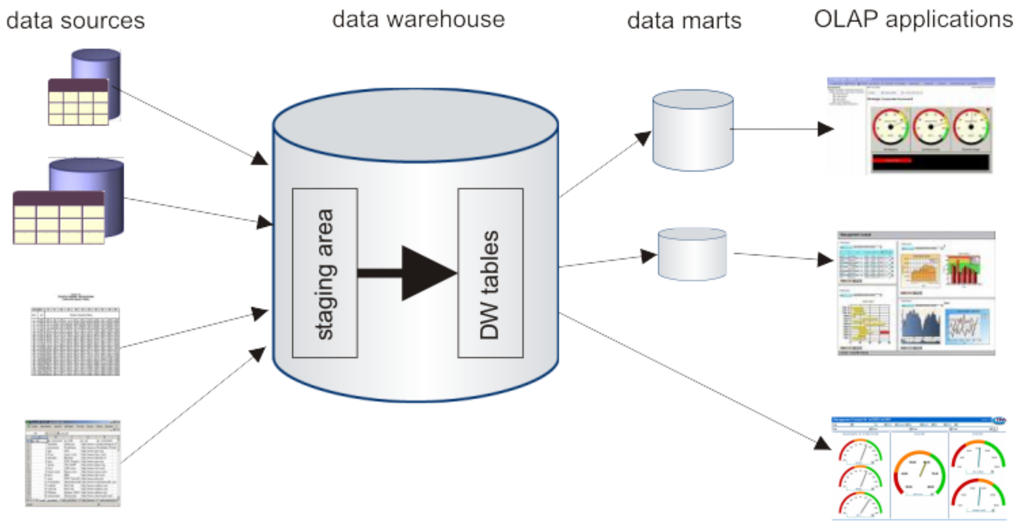

Figure 2: ELT architecture

RELATED WORK

The research areas related to the work presented in this paper include: (1) real-time data warehousing, (2) complex event processing, and (3) data stream processing.

Real-time Data Warehousing

For data warehouse systems, which address the demand for highly up-to-date data, the term real-time data warehouse has been established in the available literature, e.g., (Mohania et al., 2009; Oracle, 2010; Bateni et al., 2011; Santos et al., 2011), but it does not have much in common with the classic real-time concept. In general, a computer system is described as real-time-capable if it is guaranteed that results can be computed within fixed time intervals. In this context, the predictability, i.e., the question of when a result can be delivered is particularly interesting. The result quality remains a secondary aspect. This represents a clear contrast to the interpretation of the term real-time in the context of data warehouses. Here, the term real-time means that changes in the real world or in the modeled world are mapped to the data warehouse in a timely fashion. To achieve this, the data acquisition process and data updates as well as the query processing have to be implemented with the necessary performance. Due to the ambiguity of the term real-time, we will use the term *right-time* within this paper, in the context of our contribution.

In a real-time data warehouse (RTDW) architecture, there are two components whose performance is crucial to assuring real-time or near-real-time processing of data. The components include: ETL software and a RTDW refreshing software.

Current ETL processes, which are specialized in processing mass data, have to be adapted in order to cope with continuous updates in a right-time scenario. Logical optimization, focusing on restructuring ETL processes in order to minimize the cardinalities, is relevant here (Simitsis et al., 2005a). Simitsis et al. (2005b) propose a heuristic for searching the space of possible ETL graphs, in order to find the most efficient execution. Andzic et al. (2007) focus on designing ETL processes for high data loading frequency, for large volumes of loaded data, and for complex data processing/transformations. The authors propose to apply a layered infrastructure based on DBMS access modules, file access modules, parallel read/write modules, data processing modules and to apply partitioning, parallel processing, and pipelining to ETL processes. On the other hand, physical optimization is important, such as the insertion of artificial sorting operators (Tziovara et al., 2007) and the development of special non-blocking operators such as the mesh-join (Polyzotis et al., 2008) that support push-based processing semantics. Another research area in the context of ETL processes focuses on the incremental loading of a data warehouse: so-called incremental load jobs are derived from the already existing ETL processes (Jörg & Dessloch, 2008; Jörg & Dessloch, 2009a).

Few research results have been achieved in the area of designing ETL processes for real-time/near-real-time/active DWs. Vassiliadis & Simitsis (2009) discuss challenges in designing ETL processes for a DW of this type and propose an architecture of a near real-time DW system. This architecture is based on the ETL paradigm. A key component of the architecture is a flow regulator, associated with every data source. The flow regulator is responsible for splitting data into data processed in a real-time and data processed off line, i.e., in a standard way. The latter is stored either in main memory or on a disk, before being processed later on. In (Santos & Bernardino, 2008), real-time data are stored in staging tables having a schema structure similar to the original data warehouse. This solution uses replicas of a DW tables. The replicas are tables without any indexes or integrity constraints. The replicas are supplied with data from a continuously loaded staging area. Analytical queries work on the proper DW tables and replicas. Periodically, when the performance of the DW deteriorates, the contents of the replicas are loaded into the proper DW tables.

Thomsen et al. (2008) propose, an on-demand data loading architecture for real-time data warehouses. A novel component of this architecture is a main-memory storage, called a catalyst. Data from an ETL software can be loaded into a DW either via the catalyst or via a traditional path. User queries are accessing both the DW and the catalyst by means of views. The catalyst temporarily keeps the inserted rows locally and loads them into a DW in bulk when a predefined condition is met, e.g., the system load is below a given threshold.

Bruckner et al. (2002) present an ETL architecture that supports refreshing a DW in near real-time. The architecture is based on J2EE, where the so-called ETLets and EJB components implement ETL tasks and layers.

As mentioned earlier, the second component whose performance is crucial to assuring real-time processing is the software for refreshing the content of a DW. With this respect, there are two fundamental research issues, namely: (1) efficient refreshing DW tables and materialized views and (2) scheduling DW refreshing and analytical queries.

Materialized views (MVs) are very effective at accelerating queries especially in data warehouse scenarios. But, obviously, keeping MVs up-to-date becomes a crucial problem that is tackled by many incremental view maintenance algorithms, cf., (Gupta & Mumick, 1999). Another challenging issue concerns the selection of such a set of MVs that: (1) will be used for optimizing the greatest possible number of the most expensive queries and (2) whose maintenance will not be costly. Several research works have addressed this problem and they contributed multiple algorithms, e.g. (Gupta, 1997; Shukla et al., 1998; de Sousa & Sampaio, 1999; Theodoratos & Xu, 2004; Lawrence & Rau-Chaplin, 2006).

Updating a MV may produce inconsistent data when the MV is refreshed with data being at the same time modified by other transactions. There are a number of solutions that avoid the inconsistency. They are based among others on: (1) applying algorithms that compensate for out-of-date data in MVs, e.g., (Zhuge et al., 1995; Zhuge et al., 1996; Agrawal et al., 1997; Zhuge et al., 1999), (2) maintaining versions of MVs, where one is being refreshed while the other is being read, e.g., (Quass & Widom, 1997), and (3) using additional data structures and transactions, e.g., (Colby et al., 1996; Mumick et al., 1996).

Zhou et al. (2007) present a condensed operator in the context of maintenance of MVs. A similar operator could also work for our application but we will not go into detail here. Also, it might be possible to group updates, as it is done with many view maintenance algorithms (Salem et al., 2000).

Adelberg. et al. (1995) proposed four different algorithms for scheduling DW updates and queries, namely updates first, transactions first, split updates, and on-demand updates. The aim of these algorithms was to assure two performance quality measures: percentage of transactions missing their deadline and percentage of transactions successfully meeting their deadline, without reading stale data. Kao et al. (2003) further enhanced the scheduling algorithms by taking into account materialized view maintenance. Kim & Moon (2007) proposed a technique that permits to synchronize multiple concurrently working DW refreshing transactions and analytical queries, while ensuring serialization. This solution guarantees that the number of transactions missing their deadlines is low but the solution cannot provide data of high quality.

Thiele et al. (2009b) proposed the WINE algorithm for scheduling DW updates and analytical queries. As performance measures the authors proposed a quality of service (QoS) - reflecting a query response time and a quality of data (QoD) - reflecting data freshness. Further extensions to this scheduling approach were presented in (Thiele & Lehner, 2009). They proposed alternative scheduling strategies, namely local scheduling and global scheduling. Local scheduling exploits information on a workload of its own ETL component, whereas global scheduling leverages information on workloads of its own and all subsequent ETL components.

An algorithm for reordering transactions to improve transaction throughput by means of resource sharing is proposed in (Luo et al., 2008). The algorithm takes into account the dependencies between already running transactions and waiting transactions to be run. It maximizes sharing of resources between the transactions, e.g., cached data from disks, intermediate results of computations.

Vu & Gopalkrishnan (2009) present a scheduling algorithm for on-demand refreshing a DW tables and MVs. The algorithm assures both QoS and QoD are accomplished. The algorithm uses data versions and it schedules refreshing tasks and analytical queries so that the most up-to-date versions of data are made available for queries. The algorithm prioritizes queries by means of earliest-deadline-first.

Event Processing

Event Processing involves on-the-fly analysis of data streams. Data in these streams is seen as a series of events, which can be correlated or processed otherwise.

Complex Event Processing (CEP) engines (Wu et al., 2006; Buchmann & Koldehofe, 2009), for example, do not support complex analytical queries and advanced analytics beyond aggregation. Pure data stream management systems (Babcock et al., 2002; Abadi et al., 2003)

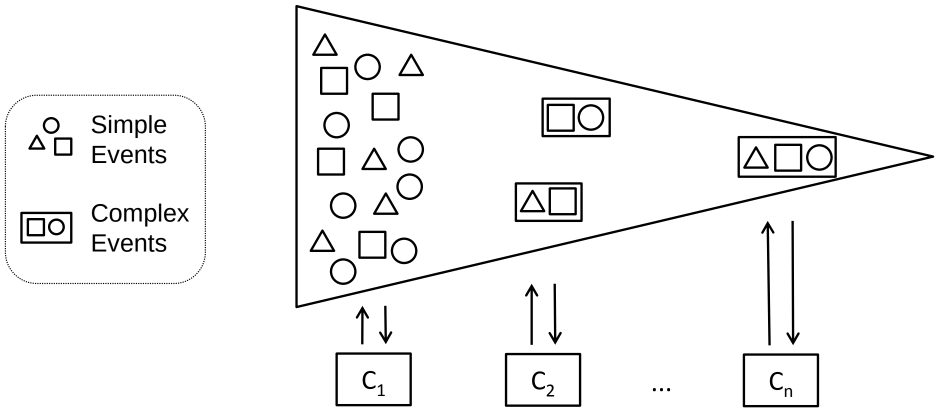

Figure 3: Complex events are combinations of simple events. Consumers (Ci) can subscribe to different levels of complexity.

can cope with high-rate input streams and rather complex aggregation queries; however, ad-hoc data analysis and advanced analytics on historic data are impossible. From the more traditional DBMS side, we have the well-known data warehouse architectures based on DB systems as well on specific platforms for the ETL tasks. However, the data transformation and refinement within ETL processes introduces a significant delay which is contrary to our right-time requirement.

Today, BI processes can benefit from analyzing continuous streams of data (e.g. a stream of temperature readings, a stream of position coordinates) (Dayal et al., 2001). Each data item from such a stream is represented as an event. An event is a lightweight entity capturing some real-world happening, including its timestamp (which can refer to the real-world time or the time the event was registered in the system). In a typical business application, there are thousands of such events per second. The traditional BI tools are too bulky, to meet the timely requirements. It is a very challenging issue to update a data warehouse multiple times per second and this issue received a lot of attention form the research community.

Event processing is suitable for continuously analyzing (multiple) streams of data and to react to significant changes therein (Chandy, 2006). The analytic logic itself is expressed in event-condition-action (ECA) rules. An ECA rule specifies a set of actions in response to a certain occurrence of events. Possible conditions include specifying a sequence of events, conditions over values in a sliding window, or simple thresholds. The possible complexity of these conditions is implementation-specific, but may be Turing-complete. The actions may of course be any action that the host system allows (e.g. showing a warning on a dashboard). Events may also be combined with ECA rules (even if they are heterogeneous) to form new, complex events, facilitating also complex analytics (see ***Figure 3***).

These tasks are often performed by data stream management systems or complex event processing engines. However, these systems are specialized to analyze only current data and are unable to combine or correlate data from a stream with historical data from a data warehouse.

Data Stream Processing

Recent research in data stream processing focused, among others things, on supplying streams of data into a data warehouse and on an efficient on-line analysis of stream data. In order to ensure on-line answers to analytical queries multiple research works proposed to apply various approximation techniques. For example, the Aqua project (Acharya et al., 1999; Acharya et al., 2000) proposed techniques of sampling an arriving data stream. Chakrabarti et al. (2001), applied wavelet-coefficient synopses of the data and based on the synopses computed approximate query answers (Chakrabarti et al., 2001). In (Condie et al., 2010) the on-line aggregation of data was supported by means of the MapReduce technique in the Hadoop framework. Gorawski and Malczok (2007) proposed a technique for parallel processing of data streams. Gorawski and Chrószcz (2011) proposed a low-level algorithm for optimizing query processing on a data stream. The algorithm uses operator partitioning in order to minimize either response time or memory usage.

Loading streamed data into a DW in real-time or near-real time is challenging for streams of high rates. The problem becomes even more challenging when MVs and indexes have to be refreshed in real-time. A solution to this problem was proposed in (Jörg & Dessloch, 2009b). They analyze anomalies caused by a DW real-time refreshing. They also show how to assure a DW consistency and avoid refreshing anomalies based the state-of-the-art ETL and database technologies. Gorawski and Marks provide a foundation for building a fault-tolerant data stream ETL architecture (Gorawski & Marks, 2006; Gorawski & Marks, 2008). The foundation is based on the impact analysis between the components of the ETL architecture and redundant component that process the same data streams in parallel. In (Kiviniemi et al., 1999) the authors developed a technique for recalculating MVs only when data freshness exceeds a given tolerance.

In summary, there is a wide spectrum of existing work and papers that address the real-time

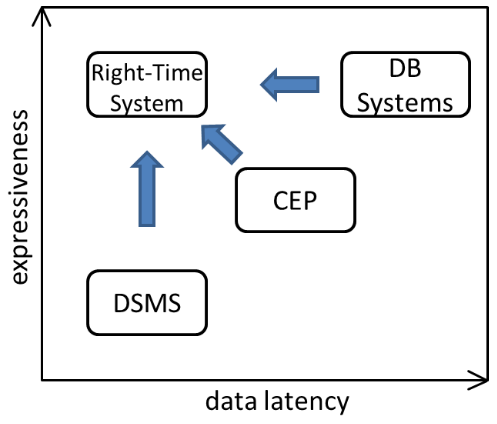

Figure 4: Expressiveness of systems vs. Latency

requirements in the context of data warehouse systems. However, no existing system category (see Figure 4) is able to provide both desired features of a Right-Time BI architecture including high expressiveness of the query language and minimal data delay. Therefore, we suggest active processing of data, taking ideas from the early work on active databases (McCarthy & Dayal, 1989; Buchmann et al., 1995). In order to provide high expressiveness of the query language and

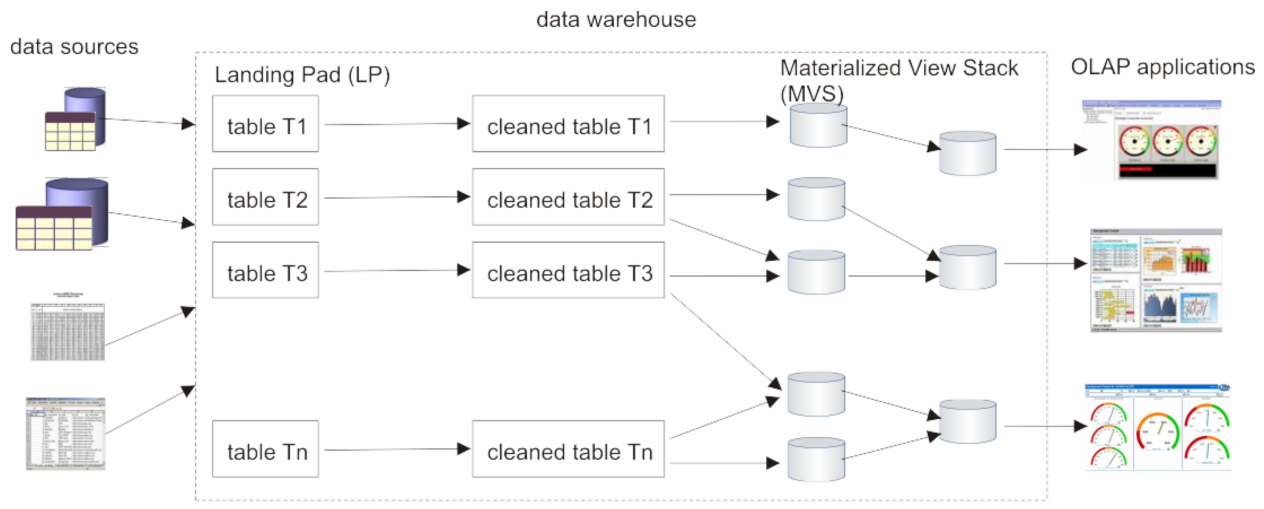

Figure 5: On-demand ELT architecture

low data latency a tailor-made Right-Time BI architecture is required. In the Right-Time BI architecture that we propose further in this paper, the aforementioned techniques of MV selection, incremental and consistent MV maintenance can be applied.

RIGHT-TIME BI ARCHITECTURE

Our work aims at achieving the two following goals. First, increasing data freshness, and second, allowing for a combined analysis of historical and stream data. To this end, we propose a basic Right-Time BI Architecture (RTBIA), as depicted in Figure 5: , which is subsequently extended with the capability of the combined analysis (cf. Figure 6). Our approach is based on two assumptions: (1) a data warehouse that is implemented by tables and the set of MVs, and (2) BI applications that access subsets of DW data and only the subsets are required to be fresh.

RTBIA is based on the ELT architecture. Data from data sources is stored in a *Landing Pad* (LP). It is a storage area that could be implemented either as a set of tables or a set of flat files. Data from LP is cleaned and loaded into proper DW tables. Based on the tables, the set of MVs is created. As shown in Figure 5: , the MVs may create a stack, called *Materialized View Stack* (MVS), where one MV is built on other MVs. OLAP applications access MVS that can also be considered as a kind of data mart.

In this architecture, MVs in the MVS are refreshed when accessed by an OLAP application (query), before the query is executed and its result delivered. The advantage of this architecture is that the system needs to refresh only these paths in MVS that are needed by the query. This allows for increasing the refreshing frequency of the DW.

Moreover, since the MVS refreshing is demand-driven, it may occur that at a given point in time, several updates exist in the landing pad that all need to access the same data object. In case these modifications mutually neutralize themselves, these redundant updates may be removed from the queue, thus saving valuable system resources.

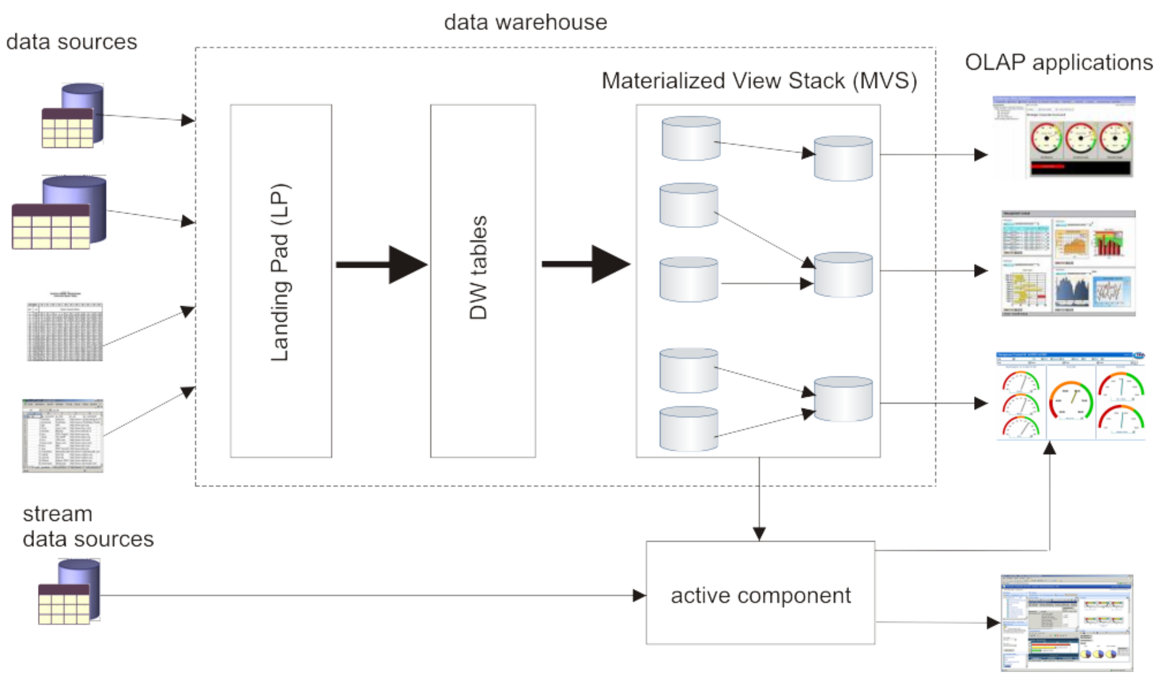

Figure 6: Querying DW and data stream

We further propose to augment this basic RTBIA to allow for live analysis of events as well as correlating historic data stored in a DW and data arriving from event data sources. Notice that, in order to be able to correlate data in the DW with stream data and obtain sound results, the DW has to contain data that is as fresh as possible. To achieve this, we add a so-called *active component*, as shown in Figure 6, into the workflow, which borrows techniques from complex event processing.

The active component can query the DW while analyzing data streams. This gives the active component the power to use results from complex analysis processes and combine it with the most up-to-date data. Since the ECA rules (cf. Section “Event Processing”) define which data are needed, this data can be cached and refreshed at convenient times. This prevents the active component from stalling due to the comparably slow loading process from the DW. Thus, the active component can detect outliers in a queried data stream and signal alerts into OLAP applications.

An alert may require immediate DW refreshing in order to perform a more advanced and detail analysis of data and to correlate the DW data with the detected outliers. Whether the DW refreshing should be triggered by the active component or an OLAP application remains an open issue, but intuitively, either of the entities could be responsible for the triggering.

Event processing pushes data to interested parties (employing a publish/subscribe mechanism). This avoids performance caveats like a busy-loop at the dashboard side. Furthermore, we can push data not only to one sink (e.g. the dashboard) but to multiple subscribers. For example, there are multiple, independent OLAP applications which are all interested in the same kind of events. Naturally, one OLAP application does not know the data needs of other OLAP applications and each of those applications would end up with its own busy-loop and data processing routine. With event processing instead, we are able to perform the analysis only once and notify all applications as soon as new data are available. In addition, this enables us to also push data directly to any other component of our redesigned, on-demand ELT architecture, with different complexity requirements (see Figure 3 and Figure 6).

Data cannot be just pushed to all the different components. Especially those components that require strict adherence to a specific schema require data transformation. Since event consumers (e.g. OLAP applications) do not know the identity of event producers, we cannot burden them with that task. In fact, this would also contract the general architecture. However, since event notifications are light-weight data structures, transformation can be done on the fly. Cilia et al. (2004) suggested an approach with a self-describing model (Cilia et al., 2004). Eugster et al. (2012) adapted this idea for typed systems, proving type safety (Eugster et al., 2012) and showing later that transformations have no measurable overhead (Freudenreich et al., 2012).

Combining the support for multiple subscribers and an arbitrary complexity level of events provides unrivaled flexibility. Different applications can subscribe for events at different complexity levels. The fine-grained combination of events to form more and more complex ones allows for a very flexible definition of the analytic task at hand. Once complexity rises beyond a manageable horizon, the analysis is complex enough that it makes sense to hand it off to the redesigned BI path.

Benchmarking the Architecture

Benchmarking ETL or ELT architectures is a challenging task. There are two aspects of the task. First, to design a benchmark that will permit to compare the performance of the standard ETL architecture with the ELT one. Both architectures can be easily made incomparable when an ETL engine is deployed on a dedicated server, while an ELT engine is deployed on a DW server. Thus, the ETL architecture will take advantage of an additional processing power of the ETL engine. Moreover, to the best of our knowledge, a standard TPC ETL/ELT benchmark has not been developed yet. Some pioneering works set up a foundation for that: Tziovara et al. (2007), Vassiliadis et al. (2007), and Simitsis et al. (2009) propose to apply the so-called butterfly-shaped ETL workflows. Wyat et al. (2009) base their benchmark on the architecture with multiple data sources (e.g., OLTP databases, XML files, text files), a staging area, and a brokerage company DW. ETL processes handle numerous data formats, populate a dimensional DW schema, as well as execute full and incremental loads.

Second, real-time data loading and a DW refreshing introduces a problem of benchmarking such an architecture, in particular - generating benchmark data streams as well as defining the set of MVs or MVS to be refreshed. To the best of our knowledge, Jędrzejczak et al. (2012) are the first to propose a benchmark for testing a refreshing job of a real-time DW. The benchmark, called *RTDW-bench*, is based on TPC-H. It permits to verify whether an already deployed RTDW can handle without any delays a transaction stream of a given arrival rate. The benchmark also includes an algorithm for finding the maximum stream arrival rate that can be handled by a RTDW without delays. *RTDW-bench* permits to determine two critical metrics. The first one, called *Critical Insert Frequency* (C_{IF}), represents a maximum average stream arrival rate below which a RTDW is capable of updating its tables without delays. The second one, called *Critical View Update Frequency* (C_{VUF}), represents a maximum average stream arrival rate below which a RTDW is capable of updating without delays its tables and refreshing all its MVs.

The *RTDW-bench* environment is composed of: (1) a data warehouse refreshed in real time, (2) a parameterized transaction stream generator, and (3) an algorithm for finding the values of C_{IF} and C_{VUF}. A DW uses the standard TPC-H benchmark schema. Based on the TPC-H tables, 30 MVs were created. The views join multiple tables and compute aggregates. They are refreshed incrementally on commit. In the benchmark, we assumed that the set of MVs is created either by a designer/administrator or by a Physical Database Design Advisor. Such advisors are supported by major commercial DBMSs (e.g., Oracle, IBM DB2, MS SQLServer). A transaction stream is characterized by a parameterized *arrival rate* and *duration time*. The stream arrival rate can be in turn parameterized by: (1) the number of transactions per a time unit and (2) the number of DML operations in a transaction.

OPEN RESEARCH PROBLEMS

The approach that we propose extends the vision of real-time/active/right-time data warehouse architectures. It still needs a substantial development. The proposed extensions, mechanisms, and architectures open new research fields and broaden the already opened ones. In the area of real-time/active/right-time data warehousing we identified **eight main open research problems** outlined below.

1. The foundation DW architectures that we proposed (cf. Figures 5 and 6) do not support querying consistent view of a DW from a given point in time, e.g., at the end of the month, week, or year. It is because a user query triggers refreshing only those MVs that are needed by the query. As a consequence, some MVs can be outdated (those not referenced by a query) and some may be up-to-date (those referenced by a query). In order to provide a consistent snapshot of the whole DW from a given point in time, the Right-Time BI internals have to be extended with temporal mechanisms, like for example (Eder et al., 2002; Malinowski & Zimanyi, 2008).
2. Since there may exist subsets of MVs in the MVS, coming from different points in time, it is evidently needed to assure the consistency of multiple dependent MVs. A naive solution of this problem would be to apply group refreshing (like in Oracle). Another solution could intelligently refresh all the MVs that contribute to a given query, taking into account query predicates. For both solutions, the system must maintain and be able to understand the dependencies between MVs.
3. Higher MV refreshing frequency results in concurrent query processing, i.e. trickle-feed updates and long-running queries have to be processed at the same time. This is of course a problem during periods of high loads in the data warehouse. To tackle the problem we have to find a reasonable balance between data currency and query performance. Some solutions to this problem, based on scheduling, have already been proposed, cf. (Thiele et al., 2009a; Thiele et al., 2009b). However, with the Right-Time BI Architecture, the problem of assuring high update rates and fast query processing is even more difficult due to the very high refreshing frequency, proactive refreshing (see research problem 4), and the existence of the MVS with multiple dependent MVs. Also the maintenance effort of the MVS should be kept at a minimum. This includes smart handling of MVs that are not referenced by any query.
4. We propose to apply on-demand refreshing of MVs driven by the current workload. Ideally, MVs refreshing should be done in a proactive manner when the system is idle. We can adopt a mix of techniques such as workload-based prediction of MVs refreshment and

demand-driven scheduling (Thiele et al., 2009a) as well as the access prediction techniques originally developed for predicting access to materialized methods, cf. (Jezierski et al., 2003; Jezierski et al., 2004; Wrembel et al., 2006). This problem opens a space for a wide spectrum of research contributions.

5. Designing MVS is another challenging task, similar to the well known view selection problem (cf. Gupta, 1999) and the optimization of multiple GROUP-BYs (cf. Chen & Narasayya, 2005). The problem of finding an optimal set of MVs is NP-complete. In the Right-Time BI Architecture we need to adjust the MVS (either automatically or semi-automatically) to the current workload, in order to keep the Stack optimal for efficient query answers and refreshing. This fact makes the problem even more challenging.
6. One of the key functionalities of RTBIA is its ability to combined querying a data stream and a DW as well as to generate alerts for OLAP applications. These yield the need for the development of a query language capable of querying and combining the results from a data stream and a DW. Moreover, a language for expressing alerting policies has to be provided. Furthermore, advanced query optimization and query execution mechanism are needed, possibly with suitable data structures.
7. Few ETL benchmarks were proposed in the research literature (cf. Related Work) and therefore there is a need for a standardized benchmark for evaluating the performance of a real-time ELT/ELT and a real-time DW. *RTDW-bench* (Jędrzejczak et al., 2012) can be further extended and standardized with the content of a data stream, the number of data streams arriving to a RTDW, the number and structure of MVs and their hierarchical dependencies in the MVS.
8. Distributed DW architectures have been proposed for increasing parallel processing of analytical queries, e.g., (Akinde et al., 2003; Furtado, 2009; Yu et al., 2011). Similar solutions can be applied to real-time/right-time DW architectures, opening new research areas like querying distributed data streams, distributed event detection and management, as well as distributed ETL/ELT over data streams.

The eight research problems are complemented by, other research problems and technological issues that have to be solved. They include among others: (1) garbage collection in the Landing Pad, (2) defining and managing data retention, (3) advanced resource governance in order to efficiently manage internal tasks in available idle time slots, including parallelization, (4) deciding on the size of updates, i.e., small vs. large refreshing batches, (5) managing indexes during DW refreshing, i.e., decisions must be taken whether to drop or not indexes for small batches, (6) efficient implementation of the Right-Time BI Architecture.

CONCLUSIONS AND FUTURE WORK

In this paper we addressed the problem of on-demand delivering fresh data into OLAP applications for the purpose of building a Right-Time Data Warehouse. In order to provide a satisfactory QoS and QoD, the Right-Time DW is being refreshed with a high frequency and only the data of interest is being refreshed. The Right-Time DW is also capable of querying at the same time a stream of constantly arriving data and the contents of the data warehouse.

In order to provide these functionalities we proposed the Right-Time BI Architecture. Its main components include: (1) the ELT framework for delivering data into a data warehouse, (2) the

MVS for implementing a data warehouse, and (3) the Active Component for querying a data stream and the data warehouse at the same time.

In this paper we also identified and outlined new research areas and extended the existing areas that are pertinent to the real-time/active/right-time data warehousing.

This paper presents the vision of a modern, next generation data warehouse system and provides the foundation for future work. Currently, at Poznań University of Technology, both architectures from Figures 2 and 5 are built for their performance comparison. Additionally, in order to test the refreshing performance of our ELT architecture, *RTDW-bench* will be applied. In the future, we will investigate the problems stated in Section "Open Research Problems".

Acknowledgements. The authors acknowledge anonymous reviewers for their constructive and thorough comments that greatly improved the quality of this paper. 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

Abadi, D. & Carney, D. & Çetintemel, U. & Cherniack, M. & Convey, C. & Erwin, C. & Galvez, E. & Hauton, M. & Maskey, A. & Rasin, A. & Singer, A. & Stonebreaker, M. & Tatbul, N. & Xing, Y. & Yan, R. & Zdonik, S. (2003). *Aurora: a data stream management system*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 666.

Acharya, S. & Gibbons, P.B. & Poosala, V. & Ramaswamy, S. (1999). *The Aqua approximate query answering system*. SIGMOD Rec., 28, pp. 574-576.

Acharya, S. & Gibbons, P.B. & Poosala, V. (2000). *Congressional samples for approximate answering of group-by queries*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 487-498.

Adelberg, B. & Garcia-Molina, H. & Kao, B. (1995). *Applying update streams in a soft real-time database system.* Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 245-256.

Agrawal, D. & Abbadi, A. & Singh, A. & Yurek, T. (1997). *Efficient View Maintenance at Data Warehouses*. SIGMOD Record, 26(2), pp. 417-427.

Akinde, M.O. & Böhlen, M.H. & Johnson, T. & Lakshmanan, L.V.S. & Srivastava, D. (2003). *Efficient OLAP Query Processing in Distributed Data Warehouses*. Information Systems, 28(1-2), pp. 111-135.

Andzic, J. & Fiore, V. & Sisto, L. (2007). *Extraction, transformation, and loading processes*. In Wrembel, R. & Koncilia, C. (eds), Data Warehouses and OLAP: Concepts, Architectures and Solutions. IGI Global, pp. 88-110.

Babcock, B. & Babu, S. & Datar, M. & Motwani, R. & Widom, J. (2002). *Models and issues in data stream systems.* Proc. of ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, pp. 1-16.

Bateni, M.H. & Golab, L, & Hajiaghayi, M.T. & Karloff, H.J. (2011). *Scheduling to Minimize Staleness and Stretch in Real-Time Data Warehouses*. Theory of Computing Systems: 49(4), pp. 757-780.

Bruckner, R.M. & List, B. & Schiefer, J. (2002). *Striving towards near real-time data integration for data warehouses.* Proc. of Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK). LNCS 2454, pp. 317-326.

Buchmann, A. & Koldehofe, B. (2009). *Complex Event Processing*. Information Technology: 51(5), pp. 241-242.

Buchmann, A.P. & Zimmermann, J. & Blakeley, J.A. & Wells, D.L. (1995). *Building an integrated active OODBMS: requirements, architecture, and design decisions*. Proc. of Int. Conf. on Data Engineering (ICDE), pp. 117-128.

Chandy, K. (2006). *Event-driven applications: Costs, benefits and design approaches*. In Gartner Application Integration and Web Services Summit.

Chakrabarti, K. & Garofalakis, M. & Rastogi, R. & Shim, K. (2001). *Approximate query processing using wavelets*. VLDB Journal, 10, pp. 199-223.

Chaudhuri, S. & Dayal, U. & Narasayya, V. (2011). *An Overview of Business Intelligence Technology*. Communications of the ACM, 54(8), pp. 88-98.

Chen, Z. & Narasayya, V. (2005). *Efficient Computation of Multiple Group By Queries*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 263-274.

Cilia, M. & Antollini, M. & Bornhövd, C. & Buchmann, A. (2004). *Dealing with Heterogeneous Data in Pub/Sub Systems: The Concept-Based Approach*. Int. Workshop on Distributed Event-Based Systems, pp. 26-31.

Colby, L.S. & Griffin, T. & Libkin, L. & Mumick, I.S. & Trickey, H. (1996). *Algorithms for Deferred View Maintenance*. SIGMOD Record, 25(2), pp. 469-480.

Condie, T. & Conway, N. & Alvaro, P. & Hellerstein, J.M. & Gerth, J. & Talbot, J. & Elmeleegy, K. & Sears, R. (2010). *Online aggregation and continuous query support in mapreduce*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 1115-1118.

Davis, J.R. (2006). Right-Time Business Intelligence: Optimizing the Business Decision Cycle. Retrieved February 2012, from http://bit.ly/Lp4njq

Dayal, U. & Hsu, M. & Ladin, R. (2001). *Business Process Coordination: State of the Art, Trends, and Open Issues*. Proc. of Int. Conf. on Very Large Data Bases (VLDB), pp. 3-13.

Eder, J. & Koncilia, C. & Morzy, T. (2002). *The COMET metamodel for temporal data warehouses*. Proc. of Conf. on Advanced Information Systems Engineering (CAiSE). LNCS 2348, pp. 83-99.

Eugster, P. & Freudenreich, T. & Frischbier, S. & Appel, S. & Buchmann, A. (2012). *Federated Objects: A Transformation Approach*. TU Darmstadt. Retrieved May 31, 2012, from http://www.dvs.tu-darmstadt.de/publications/pdf/transsem.pdf

Freudenreich, T. & Appel, S. & Frischbier, S. & Buchmann, A. (2012). *ACTrESS - Automatic Context Transformation in Event-based Software Systems*. 6th ACM International Conference on Distributed Event-Based Systems. To appear.

Furtado, P. (2009). *A Survey of Parallel and Distributed Data Warehouses*. International Journal of Data Warehousing and Mining (IJDWM), 5(2), pp. 57-77.

Golfarelli, M. & Rizzi, S. (2009). *Data Warehouse Design - Modern Principles and Methodologies*. Mc Graw-Hill.

Gorawski, M. & Chrószcz, A. (2011). *Optimization of Operator Partitions in Stream Data Warehouse*. Proc. of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP), pp. 61-65.

Gorawski, M. & Malczok, R. (2007). *Towards stream data parallel processing in spatial aggregating index*. Proc. of Int. Conf. on Parallel Processing and applied Matehmatics (PPAM), pp. 209–218.

Gorawski, M. & Marks, P. (2006). *Fault-Tolerant Distributed Stream Processing System*. DEXA Workshops, pp. 395-399.

Gorawski, M. & Marks, P. (2008). *Towards automated analysis of connections network in distributed stream processing system*. Proc. of Int. Conf. on Database Systems for Advanced Applications (DASFAA). LNCS 4947, pp. 670-677.

Gupta, H. (1997). *Selection of Views to Materialise in a Data Warehouse*. Proc. of Int. Conf. on Database Theory (ICDT). LNCS 1186, pp. 98-112.

Gupta, A. & Mumick, I.S. (1999). *Materialized Views: Techniques, Implementations, and Application*. MIT Press, 1st edition.

Jezierski, J. & Masewicz, M. & Wrembel, R. & Czejdo, B. (2003). *Designing Storage Structures for Management of Materialised Methods in Object-Oriented Databases*. Proc. of Int. Conf. on Object-Oriented Information Systems (OOIS). LNCS 2817, pp. 202-213.

Jezierski, J. & Masewicz, M. & Wrembel, R. (2004). *Prototype System for Method Materialisation and Maintenance in Object-Oriented Databases*. Proc. of ACM Symposium on Applied Computing (SAC), pp. 1323-1327.

Jędrzejczak, J. & Koszlajda, T. & Wrembel, R. (2012). *RTDW-bench: Benchmark for Testing Refreshing Performance of Real-Time Data Warehouse*. Proc. of Inf. Conf. on Database and Expert Systems Applications (DEXA). Accepted for publication.

Jörg, T. & Dessloch, S. (2008). *Towards generating ETL processes for incremental loading*. Proc. of Int. Symposium on Database Engineering and Applications (IDEAS), pp. 101-110.

Jörg, T. & Dessloch, S. (2009a). *Formalizing ETL jobs for incremental loading of data warehouses*. Prof. of Datenbanksysteme in Business, Technologie und Web (BTW), 13. Fachtagung des GI-Fachbereichs Datenbanken und Informationssysteme, pp. 327-346.

Jörg, T. & Dessloch, S. (2009b). *Near Real-Time Data Warehousing Using State-of-the-Art ETL Tools*. Int. BRITE Workshop. LNBIP 41, pp. 100-117.

Kao, B. & Lam, K. & Adelberg, B. & Cheng, R. & Lee, T.S.H. (2003). *Maintaining temporal consistency of discrete objects in soft real-time database systems.* IEEE Transactions on Computers, 52(3), pp. 373-389.

Kim, N. & Moon, S. (2007). *Concurrent view maintenance scheme for soft real-time data warehouse systems.* Journal of Information Science and Engineering, 23(3), pp. 725-741.

Kimball, R., & Caserta, J. (2004). *The data warehouse ETL toolkit*. John Wiley & Sons Inc.

Kiviniemi, J. & Wolski, A. & Pesonen, A. & Arminen, J. (1999). *Lazy Aggregates for Real-Time OLAP*. Proc. of Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK). LNCS 1676, pp. 165-172.

Lawrence, M. & Rau-Chaplin, A. (2006). *Dynamic View Selection for OLAP*. Proc. of Int. Conf. on Data Warehousing and Knowledge Discovery (DaWaK). LNCS 4081, pp. 33-44.

Luo, G. & Naughton, J.F. & Ellmann, C.J. & Watzke, M. (2008). *Transaction reordering with application to synchronized scans.* Proc. of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP), pp. 17-24.

Malinowski, E. & Zimányi, E. (2008). *Advanced Data Warehouse Design: From Conventional to Spatial and Temporal Applications*. Springer Verlag.

McCarthy, D. & Dayal, U. (1989). *The architecture of an active database management system*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 215-224.

Mohania, M.K. & Nambiar, U. & Schrefl, M. & Millist V.W. (2009). *Active and Real-Time Data Warehousing.* Encyclopedia of Database Systems. Springer, pp. 21-26.

Mumick, I.S. & Quass, D. & Mumick, B.S. (1996). *Maintenance of Data Cubes and Summary Tables in a Warehouse*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 100-111.

Oracle (2010). White Paper. Best Practices for Real-time Data Warehousing. Retrieved February 2012, from http://bit.ly/Lzcnto

Polyzotis, N. & Skiadopoulos, S. & Vassiliadis, P. & Simitsis, A. & Frantzell, N.-E. (2008). *Meshing streaming updates with persistent data in an active data warehouse.* IEEE Transactions on Knowledge and Data Engineering (TKDE), 20(7), pp. 976-991.

Quass, D. & Widom, J.(1997). *On-Line Warehouse View Maintenance*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 393-404.

Salem, K. & Beyer, K. & Lindsay, B. & Cochrane, R. (2000). *How to roll a join: asynchronous incremental view maintenance*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 129-140.

Santos, R.J. & Bernardino, J. (2008). *Real-time data warehouse loading methodology.* Proc. of Int. Symposium on Database Engineering and Applications (IDEAS), pp. 49-58.

Santos, R.J. & Bernardino, J. & Vieira, M. (2011). *24/7 Real-Time Data Warehousing: A Tool for Continuous Actionable Knowledge*. Proc. of Int. Computer Software and Applications Conf. (COMPSAC), pp. 279-288.

Shukla, A. & Deshpande, P. & Naughton, J.F. (1998). *Materialized View Selection for Multidimensional Datasets.* Proc. of Int. Conf. on Very Large Data Bases (VLDB), pp. 488-499.

Simitsis, A. & Vassiliadis, P. & Dayal, U. & Karagiannis, A. & Tziovara, V. (2009). *Benchmarking ETL workflows*. Prof. of Performance Evaluation and Benchmarking TPC Technology Conference. LNCS 5895, pp. 199-220.

Simitsis A. & Vassiliadis, P. & Sellis, T. (2005a). *Optimizing ETL Processes in Data Warehouses*. Proc. of Int. Conf. on Data Engineering (ICDE), pp. 564-575.

Simitsis, A. & Vassiliadis, P. & Sellis, T. (2005b). *State-space Optimization of ETL Workflows*. IEEE Transactions on Knowledge and Data Engineering (TKDE), 17(10), pp. 1404-1419.

de Sousa, M.F. & Sampaio, M.C. (1999). *Efficient Materialization and Use of Views in Data Warehouses*. SIGMOD Record 28(1), pp. 78-83.

Theodoratos, D. & Xu, W. (2004). *Constructing Search Space for Materialized View Selection*. Proc. of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP), pp. 112-121.

Thiele, M. & Bader, A. & Lehner, W. (2009a). *Multi-Objective Scheduling for Real-Time Data Warehouses*. In Computer Science - Research and Development, 23(3), pp. 137-151.

Thiele, M. & Fischer, U. & Lehner, W. (2009b). *Partition-based Workload Scheduling in Living Data Warehouse Environments*. Information Systems, 34(4-5), pp. 382-399.

Thiele, M. & Lehner, W. (2009). *Evaluation of load scheduling strategies for real-time data warehouse environments.* Proc. of Int. Workshop Enabling Real-Time Business Intelligence (BIRTE), pp. 84-99.

Thomsen, C. & Pedersen, T. B. & Lehner, W. (2008). *Rite: Providing on-demand data for right-time data warehousing*. Proc. of Int. Conf. on Data Engineering (ICDE), pp. 456-465.

Tziovara, V. & Vassiliadis, P. & Simitsis, A. (2007). *Deciding the physical implementation of ETL workflows*. Proc. of ACM Int. Workshop on Data Warehousing and OLAP (DOLAP), pp. 49-56.

Vassiliadis, P. & Simitsis, A. (2009). *Near real time ETL*. In Kozielski, S., & Wrembel, R. (eds), New Trends in Data Warehousing and Data Analysis. Springer Verlag, pp. 19-49.

Vassiliadis, P. & Karagiannis, A. & Tziovara, V. & Simitsis, A. (2007). *Towards a Benchmark for ETL Workflows*. Proc. of Int. Workshop on Quality in Databases (QDB), pp. 49-60.

Vu, N.H. & Gopalkrishnan, V. (2009). *Epsilon Equitable Partition: On Scheduling Data Loading and View Maintenance in Soft Real-time Data Warehouses*. Proc. of Int. Conf. on Management of Data (COMAD).

Wrembel, R. & Masewicz, M. & Jankiewicz, K. (2006). *Dynamic Method Materialization: a Framework for Optimizing Data Access via Methods*. Proc. of Inf. Conf. on Database and Expert Systems Applications (DEXA). LNCS 4080, pp. 873-882.

Wu, E. & Diao, Y. & Rizvi, S. (2006). *High-performance complex event processing over streams*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 407-418.

Wyatt, L. & Caufield, B. & Pol, D. (2009). *Principles for an ETL Benchmark*. In Nambiar, R. & Poess, M. (eds.), Performance Evaluation and Benchmarking. Springer-Verlag, pp. 183-198.

Yu, X. & Pekka, K. & Yan, Q. & Jian, W. & Keliang, Z. K. (2011). *A Hadoop based distributed loading approach to parallel data warehouses*. Proc. of ACM SIGMOD Int. Conf. on Management of Data, pp. 1091-1100.

Zhou, J. & Larson, P.-A. & Elmongui, H.G. (2007). *Lazy maintenance of materialized views*. Proc. of Int. Conf. on Very Large Data Bases (VLDB), pp. 231-242.

Zhuge, Y. & Garcia-Molina, H. & Wiener, J.L. (1996). *The Strobe Algorithms for Multi-Source Warehouse Consistency*. Proc. of Int. Conf. on Parallel and Distributed Information Systems, pp. 146-157.

Zhuge, Y. & Garcia-Molina, H. & Hammer, J. & Widom, J. (1999). *View Maintenance in a Warehousing Environment*. In Gupta, A. & Mumick, I.S. (eds.), Materialized Views: Techniques, Implementations, and Application. MIT Press, 1st edition.

Zhuge, Y. & Garcia-Molina, H. & Hammer, J. & Widom, J. (1995). *View maintenance in a warehousing environment.* SIGMOD Record, 24 (2), pp. 316-327.