Chapter 2
Near Real Time ETL

Panos Vassiliadis and Alkis Simitsis

Abstract Near real time ETL deviates from the traditional conception of data warehouse refreshment, which is performed off-line in a batch mode, and adopts the strategy of propagating changes that take place in the sources towards the data warehouse to the extent that both the sources and the warehouse can sustain the incurred workload. In this article, we review the state of the art for both conventional and near real time ETL, we discuss the background, the architecture, and the technical issues that arise in the area of near real time ETL, and we pinpoint interesting research challenges for future work.

2.1 Introduction

The demand for fresh data in data warehouses has always been a strong desideratum from the part of the users. Traditionally, the refreshment of data warehouses has been performed in an off-line fashion. In such a data warehouse setting, data are extracted from the sources, transformed, cleaned, and eventually loaded to the warehouse. This set of activities takes place during a loading window, usually during the night, to avoid overloading the source production systems with the extra workload of this workflow. Interestingly, the workload incurred by this process has been one of the fundamental reasons for the establishment of data warehouses, since the immediate propagation of the changes that take place at the sources was technically impossible, either due to the legacy nature of the sources involved or simply due to the overhead incurred, mainly for the operational source systems but also for the warehouse. In most cases, a data warehouse is typically updated every 24 hours.

Panos Vassiliadis
University of Ioannina, Ioannina, 45110, Hellas, e-mail: pvassil@cs.uoi.gr

Alkis Simitsis
Stanford University, Palo Alto, California, USA, e-mail: alkis@db.stanford.edu

Still, during the last five years business users are pushing for higher levels of freshness. To give an example, we mention a case study for mobile network traffic data, involving around 30 data flows, 10 sources, and around 2TB of data, with 3 billion rows [1]. In that case study, it is reported that user requests indicated a need for data with freshness at most 2 hours. However, business user requirements are getting more pressing as the time passes.

Nowadays, new types of sources enter into the scene. In several applications, the Web is considered as a source. In such a case, the notion of transaction at source side becomes more flexible, as the data that appear at a source web site are not always available later; therefore, if instant reaction to a change is not taken, it is possible that important information will not be gathered later, by the off-line refreshment of the warehouse. At the same time, business necessities - e.g., increasing competition, need for bigger sales, better monitoring of a customer or a goal, precise monitoring of the stock market, and so on - result in a demand for accurate reports and results based on current data and not on their status as of yesterday. Another crucial issue that questions the conventional way of thinking about ETL is the globalization of the economy and the commodity trading business. The usual process of ETL-ing the data during the night in order to have updated reports in the morning is getting more complicated if we consider that an organization's branches may be spread in places with totally different time-zones. Based on such facts, data warehouses are evolving to "active" or "live" data producers for their users, as they are starting to resemble, operate, and react as independent operational systems. In this setting, different and advanced functionality that was previously unavailable (for example, on-demand requests for information) can be accessible to the end users. For now on, the freshness is determined on a scale of minutes of delay and not of hours or a whole day. As a result, the traditional ETL processes are changing and the notion of "real-time" or "near real-time" is getting into the game. Less data are moving from the source towards the data warehouse, more frequently, and at a faster rate.

The ETL market has already made efforts to react to those new requirements (a relevant discussion can be found in subsection 3.3.) The major ETL vendors have already shipped "real time" ETL solutions with their traditional platforms. In practice, such solutions involve software packages that allow the application of light-weight transformations on-the-fly in order to minimize the time needed for the creation of specific reports. Frequently, the delay between the moment a transaction occurs at the operational site and the time the change is propagated to the target site is a few minutes, usually, five to fifteen. Such a response should be characterized more as "near real time" reaction, rather than "real time", despite how appealing and promising can the latter be in business terms.

Technically, it is not straightforward how to engineer this kind of systems, neither with respect to the overall setting and architecture, nor with respect to the algorithms and techniques employed for the ETL tasks. Similar to the case of conventional ETL processes, the commercial solutions proposed so far for near real time ETL are rather ad hoc without following a standard or a unified approach. At the same time, despite efforts targeting individual problems of this topic [29, 33, 17, 34, 25] the research community has not provided a complete approach for the design and

management of near real time ETL processes, so far. In this article, we delve into the details of the near real time ETL process and we highlight its special characteristics, with a particular point of view towards a Quality of Service (QoS) oriented architecture.

Contributions. Specifically, the main contributions of this article are as follows.

- We provide a high level discussion on the issues related to the traditional ETL processes and we denote their problems and constraints.
- We elaborate on the motives that boost the need for near real time ETL and we discuss appropriate architectures that can enable the desired functionality.
- We detail the infrastructure of near real time ETL. We present alternative topologies, issues related to parallelism and partitioning techniques, and issues concerning the communication of the different parts of the architecture.
- We provide a thorough analysis on research and engineering issues for all the phases of near real time ETL. Specifically, we discuss technical issues concerning the sources, the data processing area (the counterpart of data staging area in a near real time environment), and the warehouse, along with issues related to the flow regulators between both the source and the data processing area and the data processing area and the data warehouse.
- Finally, we discuss the state of the art in both research and industry, and we elaborate on the different industrial approaches already existing in the market.

Outline. The structure of the rest of this article is as follows. In Section 2, we give a brief overview of the techniques used and the problems of traditional ETL processes. In Section 3, we discuss the case of near real time ETL processes, we present the motivation for their use, we delve into more technical details such as their architecture and infrastructure, and also, we discuss interesting challenges and research issues for this area. In Section 4, we present the related work, and in Section 5, we conclude with a summary of the important points that have been raised.

2.2 Traditional ETL

A traditional data warehouse architecture consists of four layers: the data sources, the back-end, the global data warehouse, and the front-end. Typically, the data sources can be any of the following: On-Line Transaction Processing (OLTP) systems, legacy systems, flat files or files under any format. Modern applications have started to use other types of sources, as well, such as web pages and various kinds of documents like spreadsheets and documents in proprietary word processor formats.

The set of operations taking place in the back stage of data warehouse architecture is generally known as the *Extraction, Transformation, and Loading* (ETL) processes. ETL processes are responsible for the extraction of data from different, distributed, and often, heterogeneous data sources, their cleansing and customization in order to fit business needs and rules, their transformation in order to fit the data warehouse schema, and finally, their loading into a data warehouse.

The global data warehouse keeps a historical record of data that result from the transformation, integration, and aggregation of detailed data found in the data sources. Moreover, this layer involves datastores that contain highly aggregated data, directly derived from the global warehouse (e.g., data marts and views.) The front-end level of the data warehouse architecture consists of applications and techniques that business users use to interact with data stored in the data warehouse.

2.2.1 Operational issues and challenges

Traditionally, ETL processes deal with the following generic categories of problems:

- *Large volumes of data*. The volumes of operational data are extremely large, and incur significant data management problems in all three phases of an ETL process.
- *Data quality*. The data are not always clean and have to be cleansed.
- *Evolution of data stores*. The evolution of the sources and the data warehouse can eventually lead even to daily maintenance operations.
- *Performance issues*. The whole process has to take place within a specific time window and it is necessary to optimize its execution time. In practice, the ETL process periodically refreshes the data warehouse during idle or low-load, periods of its operation; e.g., every night. Any failures of the process must also be compensated within the specified time windows.

Additionally, in each individual phase of an ETL process several issues should be taken into consideration.

Extraction. The extraction conceptually is the simplest step, aiming at the identification of the subset of source data that should be submitted to the ETL workflow for further processing. In practice, this task is not easy, basically, due to the fact that there must be minimum interference with the software configuration at the source side. This requirement is imposed by two factors: (a) the source must suffer minimum overhead during the extraction, since other administrative activities also take place during that period, and, (b) both for technical and political reasons, administrators are quite reluctant to accept major interventions to their system's configuration.

There are four policies for the extraction of data from a data source. The naïve one suggests the processing of the whole data source in each execution of the ETL process; however, this policy is usually not practical due to the volumes of data that have to be processed. Another idea is the use of triggers at the source side; typically, though, this method is not practical due to abovementioned requirement regarding the minimum overhead at the source site, the intervention to the source's configuration and possibly, the non-applicability of this solution in case the source is of legacy technology. In practice, the two realistic policies suggest either the consideration of only the newly changed - inserted, deleted or updated - operational records (e.g., by using appropriate timestamps at the source sites) or the parsing of the log files of the

system in order to find the modified source records. In any case, this phase is quite heavy, thus, it is executed periodically when the system is idle.

Transformation & Cleaning. After their extraction from the sources, the data are transported into an intermediate storage area, where they are transformed and cleansed. That area is frequently called Data Staging Area, DSA, and physically, it can be either in a separate machine or the one used for the data warehouse.

The transformation and cleaning tasks constitute the core functionality of an ETL process. Depending on the application, different problems may exist and different kinds of transformations may be needed. The problems can be categorized as follows: (a) schema-level problems: naming and structural conflicts, including granularity differences, (b) record-level problems: duplicated or contradicting records, and consistency problems, and (c) value-level problems: several low-level technical problems such as different value representations or different interpretation of the values. To deal with such issues, the integration and transformation tasks involve a wide variety of functions, such as normalizing, denormalizing, reformatting, recalculating, summarizing, merging data from multiple sources, modifying key structures, adding an element of time, identifying default values, supplying decision commands to choose between multiple sources, and so forth.

Usually the transformation and cleaning operations are executed in a pipelining order. However, it is not always feasible to pipeline the data from one process to another without intermediate stops. On the contrary, several blocking operations may exist and the presence of temporary data stores is frequent. At the same time, it is possible that some records may not pass through some operations for several reasons, either for data quality problems or possible system failures. In such cases, these data are temporary quarantined and processed via special purpose workflows, often involving human intervention.

Loading. After the application of the appropriate transformations and cleaning operations, the data are loaded to the respective fact or dimension table of the data warehouse. There are two broad categories of solutions for the loading of data: bulk loading through a DBMS-specific utility or inserting data as a sequence of rows.

Clear performance reasons strongly suggest the former solution, due to the overheads of the parsing of the insert statements, the maintenance of logs and rollback-segments (or, the risks of their deactivation in the case of failures.) A second issue has to do with the possibility of efficiently discriminating records that are to be inserted for the first time, from records that act as updates to previously loaded data. DBMS's typically support some declarative way to deal with this problem (e.g., the MERGE command.) In addition, simple SQL commands are not sufficient since the 'open-loop-fetch' technique, where records are inserted one by one, is extremely slow for the vast volume of data to be loaded in the warehouse. A third performance issue that has to be taken into consideration by the administration team has to do with the existence of indexes, materialized views or both, defined over the warehouse relations. Every update to these relations automatically incurs the overhead of maintaining the indexes and the materialized views.

2.2.2 Variations of the traditional ETL architecture

In the majority of cases, a conventional ETL process is designed and executed, as previously described, in three major phases: Extract, Transform, and Load. However, for several business or technical reasons, other approaches are considered too. We will not elaborate on cases that involve only a subset of the ETL phases (e.g., extract and load), but we will briefly discuss a specific alternative that nowadays, gains some business interest: the case of Extract, Load, and Transform, ELT.

The crux behind the introduction of ELT solutions is twofold and based on both the feasibility of acquiring increasingly better hardware, often more powerful than needed for data warehousing purposes, and the increasing amounts of data to be handled. Hence, in the context of ELT, instead of first creating a snapshot of the operational data in the DSA and then performing the appropriate transformations, the goal is to create a snapshot of the operational data directly in the data warehouse environment, using quick batch-loading methods. Then, depending on the business needs, the administrator can decide what types of transformations to execute either on the way of data into the data marts for OLAP analysis or on a transaction-by-transaction basis in a data-mining algorithm.

The ELT approach seems beneficial in the presence of several conditions. It seems as a good solution when the operational database machines do not have enough power - while, at the same time, the data warehouse server is a more powerful machine - or when there is a slow network connection among the sources and the target warehouse. Moreover, when the population of the data warehouse is based on a single integrated version of the operational data, then having the transformation occur within the database might prove more effective, since it can take advantage of the sophisticated mechanisms that a standard RDBMS provide; e.g., the capability of issuing inserts, updates and deletes in parallel, as well as the execution of several algorithms for data mining, profiling, cleansing, and so on, from a SQL command line. Additionally, as it is stressed in a recent article [19], ELT may be more beneficial than other conventional architectures due to the following reasons: (a) it leverages RDBMS engine hardware for scalability and basically, it scales as long as the hardware and RDBMS engine can continue to scale; (b) it keeps all data in the RDBMS all the time; (c) it is parallelized according to the data set; and finally, (d) disk I/O is usually optimized at the engine level for faster throughput. Currently, most major players in the market provide ELT solutions as well; e.g., Informatica Pushdown ELT, Sunopsis ELT, Oracle Warehouse Builder, and Microsoft DTS.

Finally, an even newer trend suggests the use of ETLT systems. ETLT represents an intermediate solution between ETL and ELT, allowing the designer to use the best solution for the current need. In that case, we can classify transformations in two groups. A first group of fast, highly selective, non-blocking transformations may be executed in advance, even in streaming data, before the fast loading of the data to the warehouse area. Then, part of the incoming data can be used for fast, near real-time reporting, while the rest can be manipulated later on in a subsequent phase. Such a vision guides us to the case of near real-time ETL, which will be the focus of the rest of this article.

2.3 The case for Near Real Time ETL

2.3.1 Motivation

Traditionally, ETL processes have been responsible for populating the data warehouse both for the bulk load at the initiation of the warehouse and incrementally, throughout the operation of the warehouse in an off-line mode. Still, it appears that data warehouses have fallen victims of their success: users are no more satisfied with data that are one day old and press for fresh data -if possible, with instant reporting. This kind of request is technically challenging for various reasons. First, the source systems cannot be overloaded with the extra task of propagating data towards the warehouse. Second, it is not obvious how the active propagation of data can be implemented, especially in the presence of legacy production systems. The problem becomes worse since it is rather improbable that the software configuration of the source systems can be significantly modified to cope with the new task, due to (a) the down-time for deployment and testing, and, (b) the cost to administrate, maintain, and monitor the execution of the new environment.

The *long term vision for near real time warehousing* is to have a self-tuning architecture, where user requirements for freshness are met to the highest possible degree without disturbing the administrators' requirements for throughput and availability of their systems. Clearly, since this vision is founded over completely controversial goals, a reconciliation has to be made:

A more pragmatic approach involves a semi-automated environment, where user requests for freshness and completeness are balanced against the workload of all the involved sub-systems of the warehouse (sources, data staging area, warehouse, data marts) and a tunable, regulated flow of data is enabled to meet resource and workload thresholds set by the administrators of the involved systems.

In the rest, we will translate this vision into a list of more concrete technical goals. First, we start with the goals that concern the implementation of a near real time warehouse with a view to Quality-of-Service (QoS) characteristics:

1. *Maximum freshness of data*. We envision a near real time data warehousing environment able to serve the users with as fresh data as possible in the warehouse.
2. *Minimal overhead of the source systems*. For near real time warehousing to work, it is imperative to impose the minimum possible additional workload to the sources, which -at the same time-is sustainable by the sources.
3. *Guaranteed QoS for the warehouse operation*. The near real time warehouse administrator should be equipped with tools that allow him to guarantee service levels concerning the response time to queries posed, the throughput of all the systems involved and the freshness of data offered to the users.
4. *Controlled environment*. Since unexpected events occur throughout the daily operation of the near real time warehouse, its administrator should be equipped with the potential to respond to these events and tune the flow of data from the sources towards the warehouse with minimal effort. Any kind of optimization, prediction and automatic support of the administrator in this task is highly

valuable.

Practical considerations involve the following goals, too:

5. *Scalability in terms of sources involved, queries posed by the end users and volumes of data to be processed*. Naturally, it is expected that over time, requests for more fresh data from new sources, more available data, and a more energetic user community will have to be serviced by the warehouse. In this case, before other measures are taken (e.g., exploiting Moore's law with new hardware), the degradation of performance should be smooth.
6. *Stable interface at the warehouse side*. Apart from the smooth performance degradation due to new sources, development costs should be bounded, too. It would be convenient for developers and administrators if the warehouse would export a stable interface for its refreshment to all its source sites.
7. *Smooth upgrade of the software at the sources*. We envision a transition to a system configuration where the modification of the software configuration at the source side is minimal.

2.3.2 General architecture

We envision the general architecture of a near real time data warehouse consisting of the following elements: (a) Data Sources hosting the data production systems that populate the data warehouse, (b) an intermediate Data Processing Area (DPA) where the cleaning and transformation of the data takes place and (c) the Data Warehouse (DW). The architecture is illustrated in Figure 2.1.

Each source can be assumed to comprise a data store (legacy or conventional) and an operational data management system (e.g., an application or a DBMS, respectively.) Changes that take place at the source side have first to be identified as relevant to the ETL process and subsequently propagated towards the warehouse, which typically resides in a different host computer. For reasons that pertain to the QoS characteristics of the near real time warehouse and will be explained later, we envision that each source hosts a *Source Flow Regulator* (SFlowR) module that is responsible for the identification of relevant changes and propagates them towards the warehouse at periodic or convenient intervals, depending on the policy chosen by the administrators. As already mentioned, this period is significantly higher that the one used in the current state-of-practice and has to be carefully calculated on the basis of the source system's characteristics and the user requests for freshness.

A *Data Processing Flow Regulator* (DPFlowR) module is responsible of deciding which source is ready to transmit data. Once the records have left a certain source, an ETL workflow receives them at the intermediate data processing area. The primary role of the ETL workflow is to cleanse and transform the data in the format of the data warehouse. In principle, though, apart from these necessary cleansings and transformations, the role of the data processing area is versatile: (a)

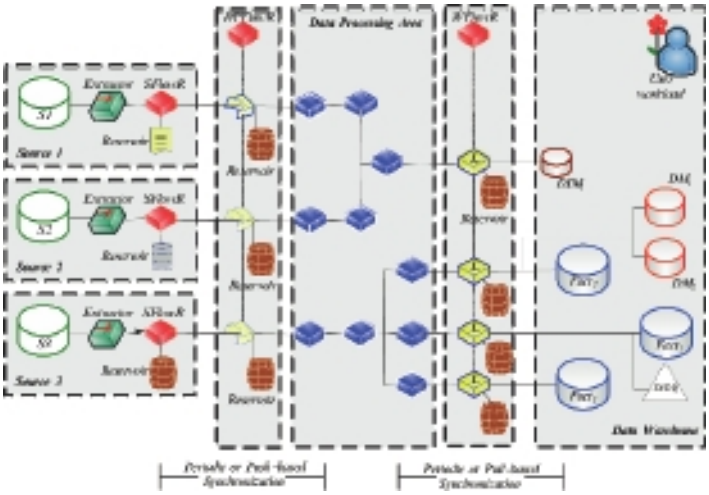

Fig. 2.1 Architecture of near real time data warehouse

it relieves the source from having to perform these tasks, (b) it acts as the regulator for the data warehouse, too (in case the warehouse cannot handle the online traffic generated by the source) and (c) it can perform various tasks such as checkpointing, summary preparation, and quality of service management. However, it is expected that a certain amount of incoming records may temporarily resort to appropriate *Reservoir* modules, so that the DPA can meet the throughput for all the workflows that are hosted there.

Once all ETL processing is over, data are ready to be loaded at the warehouse. A Warehouse Flow Regulator (WFlowR) orchestrates the propagation of data from the DPA to the warehouse based on the current workload from the part of the end-users posing queries and the QoS "contracts" for data freshness, ETL throughput and query response time. Clearly, this is a load-balancing task, which we envision to be implemented over a tunable QoS software architecture.

The Data Warehouse per se, is a quite complicated data repository. There are different categories of constructs in the warehouse, which we broadly classify as follows: (a) fact tables, containing the records of real-world events or facts, that the users are mainly interested in, (b) dimension tables, which contain reference records with information that explains the different aspects of the facts, (c) indexes of various kinds (mainly, B+-trees and bitmap indexes) which are used to speed-up query processing and (d) materialized views, which contain aggregated information that is eventually presented to the users. In the rest of our deliberations, the term 'materialized view' will be used as a useful abstraction that allows us to abstract al kinds of summaries that are computed once, stored for the users to retrieve and query and regularly updated to reflect the current status of a one or more fact tables (e.g., data marts, reports, web pages, and any possible materialized views per se.)

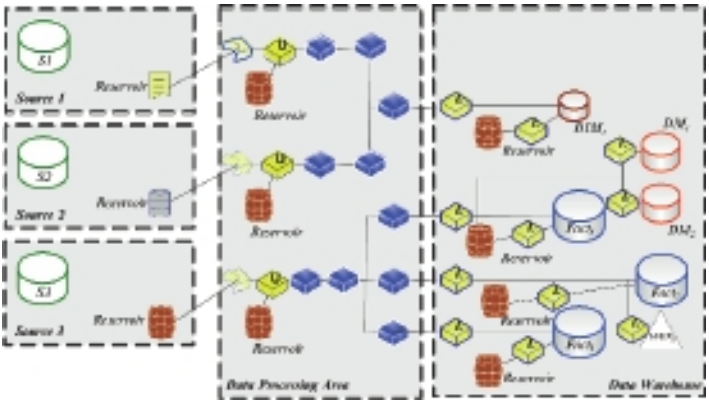

Fig. 2.2 Compensating actions for a near real time data warehouse

Each of these constructs has a clear role in a traditional ETL setting; in the case of near real time ETL, these constructs are possibly accompanied by auxiliary structures that alleviate the burden of refreshing them very frequently.

In an ideal world, all the involved systems (sources, data processing area and warehouse) would be able to process all the data within the given time windows. Clearly, this cannot be the case in practical situation, due to many possible reasons, like the high rate of user queries, the high rate of updates, the high cost of certain parts of the transformation and cleaning stage, or even the failure of a part of the overall architecture at runtime. Practically this results to the necessity of reserving parts of the propagated data for later processing. In other words, a simple selection mechanism in the flow regulators needs to decide which data will be processed by the ETL workflow in near real time and which parts will be reserved in main memory, staged at the hard disk or simply shed in order to be processed later, during an idle period of the warehouse. Similarly, a failure in the near real time processing area or the warehouse can lead to data not propagated to the warehouse on time.

These practical considerations lead to the necessity of a compensation scheme that operates in an off-line mode (much like the traditional ETL mechanisms) and completes the missing parts of data in the warehouse and the materialized views (Figure 2.2.)

2.3.3 Industrial approaches

Alternative architectures for near real time ETL has been suggested in the industrial literature. In this section, we briefly describe the alternative approaches and we pinpoint their advantages and disadvantages. For further details, we refer to [18] for an excellent review of such approaches.

2.3.3.1 (Near) Real time partition

The case of near real time ETL is discussed in an industrial book [18]. Although the description is informal and very abstract, the notion of real time partitions (RTP) seems that offer a solution to the problem of near real time ETL; however, for getting better performance several assumptions are considered too.

The key idea is to maintain two versions of the star schema representing a data warehouse. (One can imagine a virtual star schema defined by appropriate union view(-s) on top of the two star schemas.) One version should be static and the other real time, in terms of their population. Hence, using that approach, for each fact table of the data warehouse, a separate real time fact table is considered with the same characteristics as the static one - e.g., same grain and dimensionality. The real time fact table should contain only today's data that are not yet loaded to the static fact table. The static fact table is populated using conventional nightly batch loads. The main difference from the conventional method is that the real time fact table periodically populates (in short periods of time) the static fact table before being emptied.

The real time fact table should have significant performance gains, thus, the indexing in that table is minimized. In doing so, both the loading effort and the query response times are greatly benefited. However, the main advantages of this idea stem from the assumption that the whole real time fact table should fit in main memory for further fast processing. Although it may be possible to cache the fact table in memory in some cases, given that it contains data of only one day, still, this is a very ambitious assumption.

2.3.3.2 (Near) Real time ETL approaches

As usual, different alternative approaches have been proposed in the market to handle the need for freshness in a data warehouse. In what follows, we briefly mention the most prominent approaches using terminology adapted from [18] and identify their limitations.

Enterprise Application Integration, EAI. These approaches have the ability to link transactions across multiple systems through existing applications by using software and computer systems architectural principles to integrate a set of enterprise computer applications. An EAI system is a push system, not appropriate for batch transformations, whose functionality entails a set of adapter and broker components that move business transactions - in the form of messages - across the various systems in the integration network. An adapter creates and executes the messages, while a broker routes messages, based on publications and subscription rules.

The main benefit from an EAI system is fast extraction of relevant data that must be pushed towards the data warehouse. In general, an EAI solution offers great real time information access among systems, streamlines business processes, helps raise organizational efficiency, and maintains information integrity across multiple sys-

tems. Usually, it is considered as a good solution for applications demanding low latency reporting and bidirectional synchronization of dimensional data between the operational sources and the data warehouse. However, as nothing comes without a cost, they constitute extremely complex software tools, with prohibitively high development costs, especially for small and mid-sized businesses. Also, EAI implementations are time consuming, and need a lot of resources. Often, many EAI projects usually start off as point-to-point efforts, but very soon they become unmanageable as the number of applications increase.

Fast transformations via Capture - Transform - Flow (CTF) processes. This solution resembles a traditional ETL process too. CTF approaches simplify the real time transportation of data across different heterogeneous databases. CTF solutions move operational data from the sources, apply light-weight transformations, and then, stage the data in a staging area. After that, more complex transformations are applied (triggered by the insertions of data in the staging area) by microbatch ETL and the data are moved to a real time partition and from there, to static data stores in the data warehouse. CTF is a good choice for near real time reporting, with light integration needs and for those cases where core operations may share periods of low activity and due to that, they allow the realization of data synchronization with a minimal impact to the system.

Fast loading via microbatch ETL. This approach uses the idea of real time partitioning (described in section 2.3.3.1) and resembles traditional ETL processes, as the whole process is executed in batches. The substantial difference is that the frequency of batches is increased, and sometimes it gets as frequent as hourly. Several methods can be used for the extraction of data - e.g., timestamps, ETL log tables, DBMS scrapers, network sniffers, and so on. After their extraction the data are propagated to the real time partition in small batches and this process continuously runs. When the system is idle or once a day, the real time partitions populate the static parts of the data warehouse. The microbatch ETL approach is a simple approach for real-time ETL and it is appropriate for moderate volumes of data and for data warehouse systems tolerant of hourly latency. The main message it conveys, though, is mainly that dealing with new data on a record-by-record basis is not too practical and the realistic solution resolves to finding the right granule for the batch of records that must be processed each time.

On-demand reporting via Enterprise Information Integration (EII). EII is a technique for on-demand reporting. The user collects the data he needs on-demand via a virtual integration system that dispatches the appropriate queries to the underlying data provider systems and integrates the results. EII approaches use data abstraction methods to provide a single interface for viewing all the data within an organization, and a single set of structures and naming conventions to represent this data. In other words, EII applications represent a large set of heterogenous data sources as a single homogenous data source. Specifically, they offer a virtual real time data warehouse as a logical view of the current status in the OLTP systems. This virtual warehouse is delivered on-the-fly through inline transformations and it is appropriate for analysis purposes. It generates a series of (SQL) queries at the time requested, and then it applies all specified transformations to the resulting data and presents the result to

the end user. EII applications are useful for near-zero latency in real time reporting, but mostly for systems and databases containing little or no historical data.

2.3.4 The infrastructure of near real time ETL

The software architecture described in section 2.3.2 is constructed in such a way that the goals of section 2.3.1 are achieved with QoS guarantees. In this subsection, we discuss possible alternatives for the infrastructure that hosts this software architecture and facilitates the near real time refreshment of the data warehouse.

A traditional approach towards the topology of the near real time warehouse would structure it as a linear combination of tiers - in fact, as a 2-tier or 3-tier configuration. Apart from the traditional architectural configurations, it is quite natural for an endeavor of the magnitude of a near real time warehouse to opt for configurations where the architecture exploits some forms of parallelism. In this subsection, we discuss how different parallelism techniques affect the execution of ETL processes and suggest improvements in the presence of certain requirements. In general, there exist two broad categories of parallel processing with respect to the flow and volume of data: *pipelining* and *partitioning*.

In Figure 2.3, the execution of an abstract ETL process is pictorially depicted. In Figure 2.3(a), the execution is performed sequentially. In this case, only one instance of the ETL process exists. Figures 2.3(b) and 2.3(c) show the parallel execution of the process in a pipelining and a partitioning fashion, respectively. In the latter case, larger volumes of data may be handled efficiently by more than one instance of the ETL process; in fact, there are as many instances as the partitions used.

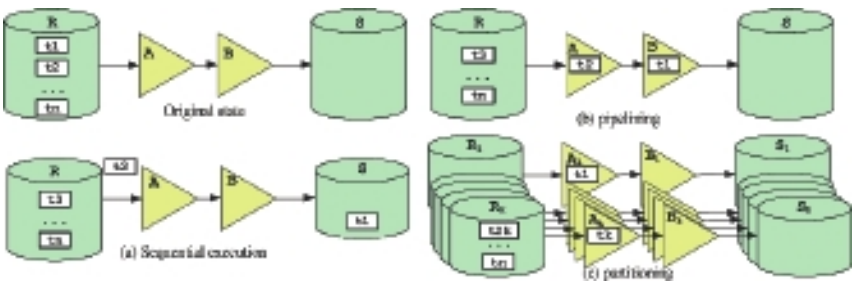

Fig. 2.3 (a) Sequential, (b) pipelining, and (c) partitioning execution of ETL processes [37]

Pipelining methods. The execution of an ETL process can be coarsely divided in three sub-processes: extraction, transformation, and loading. In pipeline parallelism, the various activities of these three sub-processes are operating simultaneously in a system with more than one processor. While the ETL process lasts, the extraction module reads data from the sources and keeps feeding a pipeline with the data it

had already read. In the meantime, the transformation module runs in another processor and it is continuously sending data to another pipeline. Similarly, the loading module, which runs in a third processor, is writing data to the target recordset. This scenario performs well for ETL processes that handle a relative small volume of data.

Partitioning methods. For large volumes of data, a different parallelism policy should be devised: the partitioning of the dataset into smaller sets. The idea is to use different instances of the ETL process for handling each partition of data. In other words, the same activity of an ETL process would run simultaneously by several processors, each processing a different partition of data. At the end of the process, the data partitions should be merged and loaded to the target recordset(s). For partitioning, many implementations have been proposed with the common goal to provide equal size partitions to facilitate the load of data to a single target. The most frequently used methods are the following. (Here, we use a terminology adopted from the DataStage tool [2], but the categorization is typical of a broader group of commercial ETL tools.)

- *Round robin partitioning*. The records are distributed among the different processing nodes in a round robin fashion: the first record goes to the first node, the second record to the second node, and so forth. This method is appropriate for resizing uneven partitions of an input data set and it is often used as the default method for the initial partitioning.
- *Random partitioning*. The records are randomly distributed across all processing nodes. This method can be used for resizing partitions, as well, and it can guarantee that each processing unit handles (near) equal-sized partitions. Moreover, this method produces results that are similar to the ones of the round robin method, but with a higher overhead than the latter due to extra processing required for the estimation of a random value for each record; this value is used as a criterion for the partitioning.
- *Hash by field partitioning*. Tuples with the same values for all hash key attributes are assigned to the same processor. Hence, related tuples are placed in the same partition. This property may be a prerequisite for a certain activity (e.g., duplicate elimination.) A similar method that requires simpler computation is the modulus partitioning, which is based on a key column modulo the number of partitions. Range partitioning is another method that places related tuples (having their keys within a specified range) in the same partition. This can be useful for preparing a dataset for a total sort.
- *Follow-the-database partitioning*. This method suggests to partition data in the same way a DBMS would partition it. Therefore, the tuples processed by the ETL process and the respective tuples of a database table would be handled by the same parallel operator (this may significantly reduce the I/O cost.) Such a method is useful for update operations, and works very well when the warehouse is partitioned too.

Two additional methods are frequently used in ETL processes to facilitate the partitioning. Although they do not intend the direct partitioning of the input data,

still they are useful in an environment consisting of subsequent activities. The goal of both methods is to efficiently pass partitioned data through a sequence of ETL activities.

- *Same partitioning*. This method does not perform a repartitioning, but it takes as inputs the partition outputs of the preceding stage. Essentially, it does not allow redistribution of data, which remains in the same processor, but, it is useful for passing partitioned data between activities.
- *Entire partitioning*. This method is appropriate when a parallel execution is desired and at the same time, all instances of an activity in all the processors should have access to the complete dataset if needed. Example application is the creation of lookup tables.

The reverse of the partitioning operation is to gather all the data together again. Typical approaches include: round robin, ordered, and sorted merge gathering. In general, this procedure is non-deterministic. Nevertheless, if order matters, a sorted merge should be favored.

Combination. In practice, a combination of pipelining and partitioning can be used to achieve maximum performance. Hence, while an activity is processing partitions of data and feeding pipelines, a subsequent activity may start operating on a certain partition before the previous activity had finished.

Several commercial ETL tools (e.g., DataStage) provide the functionality of repartitioning data between the activities of the process. This can be useful in many cases, such as when alteration of data groups is needed; e.g., in the case where data are grouped by month and a new grouping per state is needed.

2.3.5 Research and engineering issues for near real time ETL

Having discussed the alternatives for the infrastructure of a near real time data warehouse, we can now proceed to discuss the main research challenges raised for each of the main components and their overall setup. We organize this discussion on terms of the individual stages of data in the warehouse and summarize the discussion with regards to the overall setting of a near real time warehouse at the end of this section.

2.3.5.1 Technical issues concerning the Sources

There are several critical technical issues concerning the sources. Both due to political reasons [7] and due to the sensitivity of source systems, interventions to the source systems can only be minimal for the following causes.

- Sources are the production systems of the organizations, which implies that their main purpose is to facilitate the everyday transactions of the organization

with external entities; e.g., customers, providers, and so on. Overloading the source systems with ETL responsibilities might slow down the tasks that these systems are primarily intended to support to an extent that might jeopardize their primary functionality.
- Quite often, sources are legacy systems, and dealing with this fact from a software point of view means that it is practically impossible to intervene in their configuration at a significant extent. It is noteworthy that this is not a minor issue; in fact, the reason for the existence of these legacy systems is the cost and risk of their replacement with modern substitutes.

The main technical challenge at the source side is the identification of the data that must be propagated towards the data processing area at every transmission. This task has typically been referred at as *Extraction* (the 'E' in the E-T-L triplet.) We will refer to the data that are identified as changes as the *extracted delta* of the source. The important parameters of the data extraction problem are:

- *Minimal intervention to the configuration of the source system*. As already mentioned, this might not even be an option in legacy systems, but in any case, it is reasonable to anticipate that the list of available technical solutions is small.
- *Lightweight footprint of the extraction mechanism*. Due to the importance of the primary functionality of data sources, the resources spent (e.g., main memory or CPU cycles) and the overhead incurred to the source system (e.g., extra operations, locking of data) for the identification of the data that belong to the extracted delta have to be minimal.
- *Effectiveness Constraints*. A certain level of completeness might be desirable, e.g., all relevant data must be identified and included in the extracted delta. Correctness is also necessary here: no uncommitted data should be pushed towards the data processing area.
- *Efficiency Constraints*. Efficiency constraints might hold, too. For example, it is reasonable to require that the throughput of the extraction is above a certain threshold (or, in a similar problem formulation, that the task is performed within a relatively small time window, each time.) Also, it is possible that insertions and updates are identified as different as they occur and thus, separately kept at the sources, so that it is easier for the subsequent data processing to handle them appropriately.

Technically, the available solutions towards the identification of the changes that must be propagated to the warehouse are not satisfactory. The technical means we already possess for the extraction step are oriented towards the traditional data warehouse refreshment mode, where the extraction is performed off-line in rare frequency and with a relatively large time window. Specifically, inapplicable solutions involve (a) the reprocessing of the whole source as a naïve solution and (b) the comparison of the current snapshot of the source with its previous version for the extraction of deltas. On the other hand, we can also comment on some techniques that seem to be fit for the near real time ETL process, although each comes with problems:

- *Enterprise Application Integration (EAI) middleware*. In this case, the data production applications are enriched via EAI middleware and communicate with the data processing area by sending any changes that occur.
- *Log sniffing*. Log sniffing - also called log parsing or log scrapping - involves parsing the contents of a log between two timestamps and "replaying" the changes at the data processing area. Although inapplicable to sources without logging facilities, the solution sounds appealing, since the log file is an append-only file and the contention for the usage of different parts of it can be reasonably handled.
- *Triggers*. Another possibility, involves the usage of triggers in the source that are activated whenever a modification takes place in the source database. The triggers can write the modifications in a special purpose table, file or main memory construct that is processed whenever the extraction phase is activated. Still, triggers are only applicable to relational DBMSs, interfere with the setup of the source database and impose a considerable operational overhead at the source system.
- *Timestamping*. A final possibility involves adding timestamps with transaction time to all the tables of the source and extract only the ones with timestamps with greater value than the one of the last extraction. Still, this solution misses deletions (at least in its simple version) and significantly interferes with the source DBMS and possibly the source applications.

Summarizing, the problems of data extraction at the sources can be formulated as (a) an effectiveness problem and (b) an efficiency problem. The effectiveness problem calls for the construction of a mechanism that identifies changes with minimal intervention to the source's configuration and with light footprint in terms of main memory and CPU usages. The efficiency problem is an optimization problem that relates the volume of data to be propagated with a certain frequency from a data source towards the data processing area, the necessary mechanism for the task, and tries to minimize the necessary resources such that the efficiency and effectiveness constraints are respected.

A variant of the problem has to with the compensating operation of the warehouse refreshment process, which practically reverts back to the traditional problem of data extraction: in this case, the important constraint is to complete the extraction and transportation of the data within a much larger time window, in a batch mode.

2.3.5.2 Flow regulators between the source and the Data Processing Area

As already mentioned, it is imperative that the overall flow of data from the sources towards the warehouse is adapted in such a way that the source systems are not overloaded. The mechanism that we envision for the near real time extraction of data from the sources employs a global scheduler that directs the order with which sources are triggered to start the transmission of the extracted data they have collected as well as local regulators at the sources that adapt the flow of data to the

processing capacity of the source system, whenever this particular source is activated for transmission.

Simply put, the main idea involves (a) a *Data Processing Flow Regulator* (DPFlowR) module, which is responsible of deciding which source is ready to transmit data and (b) a *Source Flow Regulator* (SFlowR) module for each source, which compiles changes in blocks and propagates them towards the warehouse.

The Data Processing Flow Regulator has to decide the order of the activation of sources, the time window for the activation of each source and notify the server of the Data Processing Area for this activation (since, there is an ETL flow that will receive the newly arrived source data for further processing.) This decision possibly depends upon the availability of the source systems, their workload (that can be reported to the DPFlowR by the corresponding SFlowR for each source) and the workload of the server of the Data Processing Area.

Whenever activated, the Source Flow Regulator knows that there is a time window for the transmission of collected data. If the collected data are more than the volume that can be transmitted during the time window (due to server load, network connection or any other reasons), then there is a part of the data that will not be propagated to the warehouse during this particular transmission. If the source server is not so loaded, it is possible to perform some form of sampling for the choice of the data to be reserved. These data can be buffered either for the next transmissions over the regular operation of the near real time warehouse or for the compensation period. Also, these data can be completely discarded (if the source server is too busy to perform this kind of extra staging) and re-extracted at the compensation period.

A communication part takes place between the data processing area and the source. Several issues arise concerning, concerning the protocol of transmission (e.g., is TCP or UDP) and the completeness of transmitted data. In other words, unless other considerations arise, it is important to achieve a packet-lossless transmission. It is also noteworthy that any compression and encryption operations (that are typically encountered whenever a source communicates with the warehouse) should also be part of the decisions made by the Source Flow Regulator.

In our understanding, the sensitive part of the architecture lies in the sources, thus, the fundamental criteria for scheduling decisions concern their availability and load; nevertheless, in case the Data Processing Area server is continuously too loaded it is possible that the sensitivity of the decision must be shifted towards the DPA. Due to these possible sensitivities, we can only emphasize the importance of a Quality-of-Service approach to the system architecture. As it has probably been made obvious so far, the frequency of communication between a source and the Data Processing Area, the time window for the transmission, the volume of data to be transmitted and the scheduling protocol constitute a highly controlled mechanism (a plan or a "contract", if you like) for orchestrating the population of the Data Processing Area from the sources. The mechanism for handling any possible deviations from this plan is also part of the QoS-oriented nature of the architecture.

A second point we would like to emphasize here is the role of the flow regulation mechanism as a buffer between the data warehouse and the sources. In our opinion, it is practically improbable to ever achieve a modus operandi where committed

changes are directly propagated from the sources to the warehouse. As already explained this is due to the deep nature, architecture and technical characteristics of the sources as production systems for their organizations. Therefore, a compromise must be achieved and such a compromise requires a regulation policy that allows the sources to be relieved from the burden of feeding the warehouse with data when they are too loaded. The flow regulation mechanism implements this compromise.

From a technical viewpoint, there are three implications that result from this observation:

- There is a technical choice for the modus operandi of the propagation process which can be (a) periodic (with significantly higher frequency than the current ETL processes), (b) pull-based, with the data processing area requiring data from the sources or (c) push-based, with the sources sending extraction deltas to the data processing areas at their earliest possible convenience. Due to the sensitivity of the sources, we believe that the pull-based option is rather hard to implement; nevertheless, in all cases, precautions have to be taken to allow a source to be relieved from extra processing if it is unable to perform it at a given time point.
- A rather simple implication concerns the control of the flow regulation. We anticipate that the overhead of scheduling the whole process will be assigned to the DPA server; still, this requires some coordination with the source systems. Still, a clear research issue concerns the scheduling protocol itself (i.e., the order by which the DPA receives data from the sources along with the time window that each connection is activated.) Obviously, the scheduling problem is more complicated in the case of parallel and partitioned DPA, where multiple activation can -and have to- take place each time.
- A final implication concerns the granularity of the propagated data. There are two ways to deal with this issue: either to deal with each modification in isolation or to compile blocks of records at the source side. In the first case, whenever a data manipulation command is committed, the transportation mechanism is notified and deals with it in isolation. In the second case, nothing is ready for transportation, until a number of records is completed. Then, all records together are sent to the data processing area. Clearly, a near real time data warehouse practically has no other option than the latter (see also [17] for an experimental evaluation of the overhead incurred at the sources.)

Summarizing, the main problem for the control mechanism that facilitates the near real time propagation of extracted changes from the sources to the warehouse requires to relate (a) a scheduling protocol that regulates the order of source activations and the time window for each such activation, (b) the characteristics of the propagation (data volumes, frequency), (c) the necessary CPU and main memory resources of the DPA server, and (d) the communication protocol between sources and the DPA, in order to transfer as many data as possible to the Data Processing Area (and, ultimately, increase the data warehouse freshness) with quality of service guarantees, such as bounded latency per record packet, bounded packet losses during transmission, no starvation problems for any source, and so on.

2.3.5.3 Technical issues concerning the Data Processing Area

The Data Processing Area involves one or more servers that host workflows of transformation and cleaning activities (along with any temporary data stores) that take extracted source deltas as input and produce records ready to be stored at the warehouse fact and dimension tables.

The Data Processing Area is the equivalent of the Data Staging Area of the traditional ETL processes, where all the cleaning and transformations take place. The main technical differences of a traditional warehouse with a near real time setting are also reflected in the difference between DPA and DSA and are summarized as follows:

- In the case of near real time ETL, the frequency of execution is significantly higher, and both the data volumes to be processed as well as the time windows for the completion of the task are much smaller.
- Due to small time allowance and the pressing demands for high throughput, data staging is not always available or desirable. The same applies for costly operations, such as external sorting, that allow faster algorithms (sort-merge join or sort-based aggregations) to be employed at the DSA, in the case of traditional ETL.
- On the other hand, due to the smaller volume of data that are processed each time, many operations can be performed in main memory. This makes the memory allocation problem much more important in the case of near real time ETL.
- In the case of traditional ETL, the retention of data for further processing is performed only for data that are detected as problematic ("dirty") as well as in the case of failures. In the case of near real time ETL, data are reserved also for performance reasons, when data completeness is traded for processing throughput.

As an overall desideratum, it is also interesting to point out that a near real time ETL process is focused towards achieving high throughput at its end points, whereas a traditional ETL process is focused towards meeting the deadline of its time window for completing the processing of all the incoming tuples.

Note also that the DPA acts as a flexible buffering area: the input rate of records is not controlled by the DPA administrator, but rather, depends on the characteristics and the workload of the sources, each time; at the same time, the DPA has to guarantee a certain rate of processed records that are ready to be loaded at the warehouse whenever appropriate, necessary, or dictated by the user needs. Therefore, CPU, main memory and storage capabilities along with adaptive processing mechanism must be at hand, for handling either an overload of incoming records or a high demand for output records that can be loaded to the warehouse.

From a server configuration point of view, the DPA is a clear candidate for parallelism and partitioning. The technical options have been explained in a previous subsection, so here we would only like to point out that due to the nature of DPA as

a flexible buffer, extra processing and storage resources can frequently prove to be very helpful.

From a software architecture point of view, we can identify several interesting technical challenges. First, the question arises on how to architect the ETL activities and their communication. Clearly, intermediate staging temporary tables are not an option. A possible solution for the communication of ETL activities is to use input/output queues at the end points of each activity. Still, this solution is not without problems: for example, "a proper emptying rate for the ETL queues has to be determined. A high arrival rate compared to the configured service rate will result in instability and queue length explosion. On the contrary, a very high service rate potentially results in too many locks of the queue (resulting again in delay, contrary to what would normally be expected). It is obvious that the service rate should be close to the arrival rate in order to have both efficient service times, and as less locks as possible." [17]. Moreover, a certain degree of checkpointing might also be desirable; plugging such functionality to the above architecture while retaining throughput guarantees is not obvious. Although this solution gives a first possibility for structuring the interconnection of ETL activities in a multi-threaded environment, extending it to parallel, partitioned or both parallel and partitioned environments is not straightforward. Other options should be explored too.

The overall research problem that pertains to the DPA involves a negotiation of conflicting goals as well as some hard constraints that cannot be neglected. Specifically, one has to find a workable solution that takes into consideration: (a) the requests (or possibly "contracts" in a QoS setting) for data freshness and completeness by the end users at the warehouse, (b) the incoming rates of data that the sources can offer or sustain, (c) the ETL process that the data must go though, parts of which are simply non-replaceable (even if some cleaning is temporarily omitted, there are several transformation that simply have to be respected for schema and reference value compatibility of the produced data and the warehouse), and (d) the software and hardware configuration of the DPA. The overall goal is to satisfy as much as possible the end users, concerning the volume and freshness of the produced data (in other words: the DPA must maximize the throughput of produced data), without sacrificing any strong constraints on these values and to minimize the resources spent for this purpose. The technical decisions that must be taken, concern the following aspects:

- It is possible that the DPA must resort to the reservation of a certain amount of incoming records, so that the DPA can meet the abovementioned throughput for all the workflows that are hosted there. In this case, a first decision must be made on whether the server needs or can perform tuple reservation and a second decision must be made on how many and which tuples will be reserved. A very important architectural decision involves whether the reservation of tuples will be performed once, at the reception of data from the sources, or many times, within several activities in the ETL workflows.
- A second important problem concerns the scheduling of the ETL workflows in the DPA server. This is a classical scheduling problem at a first sight; nevertheless, complicated workflows, each with several activities are involved, requiring

scheduling policies that are both simple (and thus, quick to decide) and reasonably accurate.
- Apart from the assignment of CPU cycles to workflows and activities, a third problem concerns the allocation of main memory to workflows and activities. This is a very important decision, possibly more important than the scheduling protocol, since the problem faced in near real time ETL workflows is a main memory processing problem, to a large extent.
- Finally, given the above particularities, there is always the problem of deciding the optimal configuration of the involved ETL workflows, both at the physical and the logical level. The problem is more intense since very few results already exist for the traditional case of ETL.

Variants of the above problems do exist, too. For example, instead of a global scheduling policy, one could think of eddie-like self-adaptive activities that take local decisions. This can be generalized to the overall architecture of the near real data warehouse. Micro- and macro-economic models can also be employed [34]. Another clear variant of the overall problem, with significant importance for the design of the DPA, involves fixing some of the desired measures (source rates, throughput) and trying to determine the best possible configuration for the DPA server. Finally, it should be stressed that the research community is not equipped with appropriate benchmarks to experiment with; this creates problems both for the validity of experimental results and for the repeatability of the experiments.

2.3.5.4 Flow regulation between the Data Processing Area and the data warehouse

Once data pass through the transformation and cleaning process in the data processing area they have to be loaded at the data warehouse. In the traditional ETL setting, this problem had a straightforward solution: all data would pass through the appropriate data loader, which is a vendor specific tool that takes care of the loading of data in a manner that is more efficient than any other alternative. Apart from the dimension and fact tables of the warehouse, special care is taken for any indexes or materialized views, too. In the case of near real time ETL, the luxury of the off-line idle warehouse that is loaded with data is no longer available and a new architecture must be investigated mainly due to the fact that the warehouse server is used by the end users for querying purposes at the same time that data are loaded to the queried tables or materialized views.

We envision an architecture where the flow of data from the DPA to the warehouse is regulated by a *data Warehouse Flow Regulator* (WFlowR) so that (a) the warehouse receives data according to the demands of users for freshness and completeness, (b) the querying processes that are initiated by the users are not significantly delayed and (c) no data are lost at the DPA side due to the overflow of produced tuples as compared to the availability of loading at the warehouse side. The ultimate desideratum is to maximize the satisfaction of the above antagonizing

goals while spending as few system resources as possible in terms of main memory, hard disk and CPU cycles.

The technical issues that arise are similar to the case of flow regulation between sources and the DPA and can be listed as follows:

- Once again, we need to take care of communication protocols, compression and encryption as well as a decision on the granularity of transmitted data.
- A scheduling policy concerning the order of data transmission is important. More important than that, though, is the issue of modus operandi, which offers the following options: (a) periodic, (b) push-based, propagating blocks of produced records to the warehouse as they come and (c) pull-based, where the scheduler decides to push data towards the warehouse given a request by the users or a predicted decrease of the warehouse load. As already mentioned, we anticipate the DPA to be the means to alleviate the possible overload of the sources and the warehouse and therefore, we find the second alternative less likely to be of practical use in warehouses with high load. On the contrary, this is a really useful solution for underused warehouses.
- Again, since the warehouse load, the user requests, and the tuple production are antagonizing goals, it is possible to perform a certain amount of tuple reservation in an attempt to produce a solution that respects all the possible constraints and maximizes a satisfaction function that combines the above goals.

2.3.5.5 Technical issues concerning the Warehouse

The data warehouse side of the ETL process is responsible for refreshing the contents of the warehouse with newly produced data that come all the way from the sources. This task is called *Loading* (the 'L' in the E-T-L triplet) and comprises the following sub-tasks, in the following order:

- Loading of the dimension tables with lookup, reference values
- Loading of the fact tables with factual data
- Maintenance of indexes and materialized views

In a traditional ETL environment, this task is performed via vendor specific tools called loaders, when the data warehouse is idle -or even off-line. To this day, vendor-specific loaders are the fastest way to implement the loading of data to a relational database. Also, dropping and recreating the indexes is sometimes of comparable time and efficiency with respect to storage and querying than incrementally maintaining them. Unfortunately, the luxury of an idle warehouse is not present in a near real time warehouse, where the loading has to take place concurrently with the answering of queries posed by the end users. Therefore, the near real time warehouse has to find an equilibrium between two antagonizing goals, the near real time refreshment of its contents and the on-line answering of queries posed by the end users. This antagonism has at least two facets:

- A contention for server resources, such as main memory and CPU cycles, that has a clear impact on the performance (at least as far as the end users perceive it.)
- A contention for database resources, such as transaction locks, in the case of strict isolation levels (although, we anticipate that the user requests for consistency of the data they receive is low - and in any case, contradicting the requirement for as fresh data as possible.)

The technical topics that result from this problem concern the hardware configuration of the warehouse, the design of the data warehouse, and the implementation of a loading mechanism that maximizes data freshness without delaying user queries above a certain tolerance level. Specifically, these topics can be detailed as follows.

A first concern involves the hardware configuration of a data warehouse. Although warehouses are not as rigid hardware environments as sources are, still, it is not straightforward how to migrate an existing warehouse to a highly parallel and partitioned architecture that is probably needed for a near real time warehouse. Moreover, the scientific community is not in possession of a cost model that can relate the extent of parallelism and partitioning required to sustain a near real time ETL process along with on-line querying.

A second concern, with a clear research challenge involves the design of the data warehouse. For the moment, a warehouse is built on the basis of a star or snowflake schema, combined with bitmap of B+ tree indexes for performance reasons. On top of these constructs, data marts, reports, web pages and materialized views are also maintained by the refreshment process, once data have been loaded to the fact tables. Remember that so far, in the context of our deliberations, we have abstracted all these constructs as materialized views. The research question that arises asks whether new kind of schema structures, along with novel kinds of indexes and even, novel kinds of materialized views are necessary for the implementation of near real time data warehousing.

A third topic involves a scheduling problem: in what order do we schedule the loading of the data warehouse tables, and what time windows are allowed for each table, every time we activate a loading process? Naturally, this problem spans both traditional relations and data structures as well as the aforementioned novel structures that might prove useful for near real time warehousing. A parameter of the problem that might possibly complicate the solution has to do with the refreshment strategy: different choices have to be made in the case where the strategy is periodic as opposed to the case where the strategy is pull-based. Obviously, in the case of partitioning and parallelism, the problem is complicated (although the available computing power is much more) since we need to schedule the simultaneous loading of different constructs or parts of them.

A final topic of research concerns both the warehouse and the flow regulation towards it and has to do with the monitoring of the current workload and the forecast of the forthcoming workload in the very near future. Monitoring is a problem per se, since it has to trade off accuracy with simplicity and a small footprint. Predicting the near future in terms of user load is even harder, to a large extent due to the irregular nature of the behavior of the users. Still, any form of scheduling for the

ETL process requires some estimation for the forthcoming user load and thus, a reasonable accurate such estimation is valuable.

2.3.6 Summing up

Coming back to the big picture, the main idea is that the data processing area has to be regulated so that the user request for fresh and complete data is balanced against the computational needs of (a) the source systems (due to their regular workload and the non-negotiable nature of their configuration and performance), (b) the data processing area (due to the rigid necessity for transforming, and sometimes, cleaning source data to a schema and value set that is acceptable for the warehouse), and (c) the warehouse (due to the on-line support of user queries.) Both the user requests and the computational needs of the involved systems can be formulated in part as non-negotiable hard constraints and in part as negotiable soft constraints, accompanied by a satisfaction function that has to be maximized. This formulation can lead to a quality of service problem, provided that we have the means to relate the involved parameters and measures via a realistic cost model.

The overall problem for near real time data warehousing is reduced to the design and regulation of such an environment in terms of (a) hardware configuration, (b) software architecture and, given the above, (c) resource allocation so that an acceptable compromise can be found for all these antagonizing goals.

2.4 Related Work

Terminological issues

We believe it is worthwhile to spend some lines to discuss terminological issues. We have chosen to use the term near real time warehousing, despite the fact that different variants already exist for this name.

Why not *active* ETL or data warehousing? In [17], the term *active data warehousing* is used in the sense of what we now call near real time data warehousing -i.e., it refers to an environment "where data warehouses are updated as frequently as possible, due to the high demands of users for fresh data". Some years before, the term *active data warehouse* has also been used in [29, 33] to refer to a warehouse that is enriched with ECA rules that provide automatic reporting facilities whenever specific events occur and conditions are met.

We have decided not to use the term, both due to its relevance with active databases and due to its previous usage in a different context. The relevance to active databases is particularly important, since it carries the connotations of ECA rules and triggers as well as a tuple-level execution model for the triggering of the ETL process (as opposed to our understanding for a block-based execution model.)

Why not *real time* ETL or data warehousing? For obvious reasons, industry favors the term *real time* warehousing as it comes with the connotation of instant propagation of source changes to the warehouse [4, 28, 40]. As already mentioned, in this article, we take a more pragmatic approach, where there is a specified frequency for the propagation of changes to the warehouse in blocks of tuples. A second, important reason is that the term *real time* comes with hidden semantics in computer science: scheduling with "real time" constraints for successful completion before a specified deadline. Databases have also been related to such deadline constraints during the '90s, overloading the term even more [16]. Therefore, we decided to avoid this term, too.

Why *near real time* ETL? We believe that the term is the closest possible to the abovementioned industrial terminology and reflects quite accurately the meaning of the discussed concept. The term has also been used by a large industrial vendor (Oracle) in the past.

At the same time, we would also like to point out that other terms are available. Thiele et al [34] use the term *living data warehouse* environments. Oracle also uses the term *on-time data warehousing*. We find both these terms excellent candidates, too.

Traditional ETL

Despite the fact that the importance of this problem has already been recognized by industrial needs for more than two decades, only recently, with the turning of the century, research dealt with the challenging problem of ETL processes. Initial research efforts regarding traditional ETL processes have mainly focused on modeling and design issues. Several alternatives have been proposed for the conceptual design of ETL processes that use different formalisms and design methods as, for example, UML [35, 20], Semantic Web [31, 32], whilst some other efforts follows their own approach [39]. However, so far there is not a clear winner, since the admission of a standard unified method is not yet a fact. Additionally, logical models for ETL processes have been proposed too, e.g., [38].

Later on, during the last five years, research has dealt with challenges beyond the modeling of ETL processes. Some efforts towards the optimization of ETL processes have been proposed [30, 36], along with efforts related to individual operators, such as the DataMapper [5]. Although, we find research work on data cleaning [12, 11, 27], data fusion [3, 22], and schema mappings [13, 26] related to ETL processes, the work proposed on that fields originally was not directly connected to the ETL technology.

Apart from research efforts, currently, there is a plethora of ETL tools available in the market. All major database vendors provide ETL solutions and in fact, they practically ship ETL tools with their database software 'at no extra charge' [14, 21, 24]. Of course, there are plenty of other ETL tools by independent vendors, such as Informatica [15]. The former three tools have the benefit of the minimum cost, because they are shipped with the database, while ETL tools from the latter category

have the benefit to aim at complex and deep solutions not envisioned by the generic products.

Efforts related to near real time ETL

Real time reporting. In [29, 33] the authors, discuss an architecture for near-real time reporting, which they call *active warehousing*. The active warehouse includes a rule-based engine at the warehouse side that is responsible of detecting whether any Event-Condition-Action (ECA) rule, registered by the user needs to be fired. The ECA rules that are registered by the user serve as the mechanism via which reports are generated to the user whenever certain events occur and specific conditions are met. The events are mostly temporal events or executions of methods at the OLTP systems. The action part of the rules can be either the local analysis of data at the warehouse or the combined analysis of warehouse with source data (obtained via a request to the OLTP system.)

The core of the approach lies in the fact that the warehouse is enriched with a rule-based engine that triggers the appropriate rules whenever certain phenomena take place at the data. Coarsely speaking, the user registers a sequence of (cube-based) reports that he/she wants to be generated whenever the data fulfill specific conditions; e.g., a certain trend is detected in the most current data.

There are significant differences with the near-real time architecture we discuss in this article. In [29, 33], the extraction is assumed to take place periodically and the loading is performed in an off-line fashion. In other words, the ETL part of warehousing is downplayed and the architecture mostly resembles an Enterprise Information Integration environment for real time reporting. Also, to the best of our knowledge, the presented papers are not accompanied by any published experimental results on the actual behavior of an active warehouse with respect to its effectiveness or efficiency.

Data Warehouse Partitioning and Replication. Node partitioned data warehouses have been investigated to a large extent by Furtado [9, 8, 10]. The main idea of node-partitioned warehousing is to employ a cluster of low cost interconnected computers as the hardware platform over which the warehouse is deployed. The cluster is a shared-nothing architecture of computers, each with its own storage devices and data are distributed to the nodes so as to parallelize the processing of user queries. The system configuration involves neither too expensive specialized mainframe servers, nor a parallel configuration with specialized connections among the nodes.

The main problems of node-partitioned warehousing are the placement of data to nodes, and the processing of queries given that placement. Since data warehouses comprise dimension tables that come both in small and large sizes, as well as facts of high volume, Furtado proposes a scheme where small dimensions are replicated in several nodes, large dimensions are partitioned on the basis of their primary keys and facts are partitioned on the basis of their workload, with the overall goal to balance query processing and data transfer in the cluster of nodes. Furtado has extensively experimented with alternative configurations for the placement of data in nodes.

Also, replication issues have been considered, to increase the fault-tolerance of the partitioned data warehouse.

Load Balancing. The paper [34] by Thiele et al, deals with the problem of managing the workload of the warehouse in a near real time warehouse (called real time warehouse by the authors.) The problem at the warehouse side is that data modifications arrive simultaneously with user queries, resulting in an antagonism for computational resources.

The authors discuss a load-balancing mechanism that schedules the execution of query and update transactions according to the preferences of the users. Since the antagonism between user queries and updates practically reflects the inherent conflict between the user requirement for data freshness and the user requirement for low response times, for each query the users pose, they have to specify the extent to which they are willing to trade off these two measures (which are also called Quality of Data and Quality of Service by the authors.) The scheduling algorithm, called WINE (Workload Balancing by Election), proposes a two-level scheduling scheme. The algorithm takes as input two queues, one for the queries and one for the updates and starts by deciding whether a query or an update will be processed. Since each query is annotated by a score for freshness and a score for fast processing, for each of these two categories, all the scores in the query queue are summed; the largest score decides whether a query or an update will take place. Then, once this decision has been made, a second decision must be made with respect to which transaction in the appropriate queue will be scheduled for processing.

Concerning the query queue, the system tries to prioritize queries with a preference towards low response time; at the same time, this also helps queries with higher preference for completeness to take advantage of any updates that concern them in the meantime. Actions against starvation are also taken by appropriately adjusting the QoS values for delayed query transactions. Concerning the update queue, updates that are related to queries that are close to the head of the query queue (thus, are close to been executed) are also prioritized. This is achieved by appropriately adjusting an importance weight for each update, according to their relevance to upcoming queries. Specific measures are taken to hold the consistency of the warehouse with respect to the order in which they arrive at the warehouse.

To the best of our knowledge, the paper is the only effort towards the load balancing of a near real time warehouse. Of course, the focus of the paper is clearly on the loading side, without any care for the extraction, transformation or cleaning part of the process. From our point of view this is a benefit of a layering and the near-real time nature of the architecture that we propose. We believe that the organization of updates in blocks and the isolation of the different concerns (Extract - Transform - Load) in different systems allow the derivation of useful algorithms for each task.

System choices. In [17], the authors propose a framework for the implementation of near real time warehouse (called "active" data warehousing by the authors), with the goals of: (a) minimal changes in the software configuration of the source, (b) minimal overhead for the source due to the "active" nature of data propagation, (c) the possibility of smoothly regulating the overall configuration of the environment in a principled way.

The paper was built upon the idea of separating the extraction, the transformation and cleaning and the loading tasks. In terms of system architecture, the ETL workflow is structured as a network of ETL activities, also called *ETL queues*, each pipelining blocks of tuples to its subsequent activities, once its filtering or transformation processing is completed. In order to perform this task, each ETL activity checks its queue (e.g., in a periodic fashion) to see whether data wait to be processed. Then, it picks a block of records, performs the processing and forwards them to the next stage. Queue theory is employed for the prediction of the performance and the tuning of the operation of the overall refreshment process. The extraction part in the paper involved ISAM sources and a simple method for extracting changes in legacy source systems, without much intervention to the system configuration was also demonstrated. The loading at the warehousing was performed via web services that received incoming blocks for the target tables and performed the loading.

The experiments proved that this architectural approach provides both minimum performance overhead at the source and the possibility of regulating the flow towards the warehouse. The experiments also proved that organizing the changes in blocks results in significant improvement in the throughput of the system and the avoidance of overloading of the sources.

The paper lacks a deep study of different activities involved in ETL scenarios (i.e., the activities employed were filters and aggregators, without a wider experimentation in terms of activity complexity.) Also, although the usage of queue theory seemed to be successful for the estimation of the processing rates of the ETL workflow, it has not been tested in workflows of arbitrary complexity (which hides the possibility of bringing queue theory to its limits.) Finally, web services are quite heavy, although they present a reliable (syntactically and operationally) and thus promising middleware, for the interoperability of the different parts of the system. Despite these shortcomings, the experimental setup and the architectural choices give a good insight to the internals of a near real time data warehouse and justify, to a broad extent, our previous discussion on the architecture of a near real time data warehouse.

Related areas

Another area related to our approach is the one of **active databases**. In particular, if conventional systems (rather than legacy ones) are employed, one might argue that the usage of triggers [6] could facilitate the on-line population of the warehouse. Still, related material suggests that triggers are not quite suitable for our purpose, since they can (a) slow down the source system and (b) require changes to the database configuration [4]. In [23] it is also stated that capture mechanisms at the data layer such as triggers have either a prohibitively large performance impact on the operational system.

2.5 Conclusions

The topic of this article has been near real time ETL. Our intention has been to sketch the big picture of providing end users with data that are clean, reconciled (and therefore, useful) while being as fresh as possible, at the same time, without compromising the availability or throughput of the source systems or the data warehouse. We have discussed how such a configuration is different from the setting of an ETL process in a traditional warehouse. We have sketched the high level architecture of such an environment and emphasized the QoS based characteristics we believe it should have. Moreover, we have discussed research topics that pertain to each of the components of the near real time warehouse, specifically, the sources, the data processing area, and the warehouse, along with issues related to the flow regulators that regulate the flow of data under data quality and system overhead "contracts".

Clearly, the importance, complexity and criticality of such an environment make near real time warehousing a significant topic of research and practice; therefore, we conclude with the hope that the abovementioned issues will be addressed in a principled manner in the future by both the industry and the academia.

References

1. J. Adzic and V. Fiore. Data Warehouse Population Platform. In *DMDW*, 2003.
2. Ascential Software Corporation. DataStage Enterprise Edition: Parallel Job Developer's Guide. In *Version 7.5, Part No. 00D-023DS75*, 2004.
3. J. Bleiholder, K. Draba, and F. Naumann. FuSem - Exploring Different Semantics of Data Fusion. In *VLDB*, pages 1350–1353, 2007.
4. D. Burleson. New Developments in Oracle Data Warehousing. In *the Web, available at: http://dba-oracle.com/oracle_news/2004_4_22_burleson.htm*, 2004.
5. P. J. F. Carreira, H. Galhardas, J. Pereira, and A. Lopes. Data Mapper: An Operator for Expressing One-to-Many Data Transformations. In *DaWaK*, pages 136–145, 2005.
6. S. Ceri and J. Widom. Deriving Production Rules for Incremental View Maintenance. In *VLDB*, pages 577–589, 1991.
7. M. Demarest. The Politics of Data Warehousing. In *the Web, available at: http://www.noumenal.com/marc/dwpoly.html*, June 1997.
8. P. Furtado. Experimental Evidence on Partitioning in Parallel Data Warehouses. In *DOLAP*, pages 23–30, 2004.
9. P. Furtado. Workload-Based Placement and Join Processing in Node-Partitioned Data Warehouses. In *DaWaK*, pages 38–47, 2004.
10. P. Furtado. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*, chapter Efficient and Robust Node-Partitioned Data Warehouses. IRM Press (Idea Group), January 2007.
11. H. Galhardas, D. Florescu, D. Shasha, and E. Simon. AJAX: An Extensible Data Cleaning Tool. In *SIGMOD Conference*, page 590, 2000.
12. H. Galhardas, D. Florescu, D. Shasha, E. Simon, and C.-A. Saita. Declarative Data Cleaning: Language, Model, and Algorithms. In *VLDB*, pages 371–380, 2001.
13. L. M. Haas, M. A. Hernández, H. Ho, L. Popa, and M. Roth. Clio Grows Up: from Research Prototype to Industrial Tool. In *SIGMOD Conference*, pages 805–810, 2005.
14. IBM. IBM Data Warehouse Manager. In *the Web, available at: http://www-3.ibm.com/software/data/db2/datawarehouse/*, 2005.

15. Informatica. PowerCenter. In *the Web, available at: http://www.informatica.com/products/powercenter/*, 2005.
16. B. Kao and H. Garcia-Molina. An Overview of Real-Time Database Systems. In Springer-Verlag, editor, *Proceedings of NATO Advanced Study Institute on Real-Time Computing. Available at: http://dbpubs.stanford.edu/pub/1993-6*, October 9 1992.
17. A. Karakasidis, P. Vassiliadis, and E. Pitoura. ETL queues for active data warehousing. In *IQIS*, pages 28–39, 2005.
18. R. Kimball and J. Caserta. *The Data Warehouse ETL Toolkit (chapter 11)*. Wiley Publishing, Inc., 2004.
19. D. E. Linstedt. ETL, ELT - Challenges and Metadata. In *the Web, available at: http://www.b-eye-network.com/blogs/linstedt/archives/2006/12/etl_elt_challen.php*, December 2006.
20. S. Luján-Mora, P. Vassiliadis, and J. Trujillo. Data Mapping Diagrams for Data Warehouse Design with UML. In *ER*, pages 191–204, 2004.
21. Microsoft. Data Transformation Services. In *the Web, available at: http://www.microsoft.com/sql/prodinfo/features/*, 2005.
22. F. Naumann, A. Bilke, J. Bleiholder, and M. Weis. Data Fusion in Three Steps: Resolving Schema, Tuple, and Value Inconsistencies. *IEEE Data Eng. Bull.*, 29(2):21–31, 2006.
23. Oracle. On-Time Data Warehousing with Oracle10g - Information at the Speed of your Business. In *the Web, available at: http://www.oracle.com/technology/products/bi/pdf/10gr1_-twp_bi_ontime_etl.pd*, August 2003.
24. Oracle. Oracle Database Data Warehousing Guide 11g Release 1 (11.1). In *the Web, available at: http://www.oracle.com/pls/db111/portal.portal_db?selected=6*, September 2007.
25. N. Polyzotis, S. Skiadopoulos, P. Vassiliadis, A. Simitsis, and N. Frantzell. Meshing Streaming Updates with Persistent Data in an Active Data Warehouse. *IEEE Trans. Knowl. Data Eng.*, 20(7), 2008.
26. E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *VLDB J.*, 10(4):334–350, 2001.
27. V. Raman and J. M. Hellerstein. Potter's Wheel: An Interactive Data Cleaning System. In *VLDB*, pages 381–390, 2001.
28. M. Rittman. Implementing Real-Time Data Warehousing Using Oracle 10g. In *the Web, available at: http://www.dbazine.com/datawarehouse/dw-articles/rittman5*, 2006.
29. M. Schrefl and T. Thalhammer. On Making Data Warehouses Active. In *DaWaK*, pages 34–46, 2000.
30. A. Simitsis, P. Vassiliadis, and T. K. Sellis. State-Space Optimization of ETL Workflows. *IEEE Trans. Knowl. Data Eng.*, 17(10):1404–1419, 2005.
31. D. Skoutas and A. Simitsis. Designing ETL Processes Using Semantic Web Technologies. In *DOLAP*, pages 67–74, 2006.
32. D. Skoutas and A. Simitsis. Ontology-Based Conceptual Design of ETL Processes for Both Structured and Semi-Structured Data. *Int. J. Semantic Web Inf. Syst.*, 3(4):1–24, 2007.
33. T. Thalhammer, M. Schrefl, and M. K. Mohania. Active Data Warehouses: Complementing OLAP with Analysis Rules. *Data Knowl. Eng.*, 39(3):241–269, 2001.
34. M. Thiele, U. Fischer, and W. Lehner. Partition-based Workload Scheduling in Living Data Warehouse Environments. In *DOLAP*, pages 57–64, 2007.
35. J. Trujillo and S. Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *ER*, pages 307–320, 2003.
36. V. Tziovara, P. Vassiliadis, and A. Simitsis. Deciding the Physical Implementation of ETL Workflows. In *DOLAP*, pages 49–56, 2007.
37. P. Vassiliadis, A. Karagiannis, V. Tziovara, and A. Simitsis. Towards a Benchmark for ETL Workflows. In *QDB*, pages 49–60, 2007.
38. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Inf. Syst.*, 30(7):492–525, 2005.
39. P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In *DOLAP*, pages 14–21, 2002.
40. C. White. Intelligent Business Strategies: Real-Time Data Warehousing Heats Up. In *DM Review. Available at: http://www.dmreview.com/article_sub.cfm?articleId=5570*, August 2002.