On the evaluation of data freshness in data integration systems

Verónika Peralta – Mokrane Bouzeghoub

Laboratoire PRiSM, Université de Versailles
45, avenue des Etats-Unis
78035 Versailles cedex, FRANCE
Veronika.Peralta@prism.uvsq.fr - Mokrane.Bouzeghoub@prism.uvsq.fr

ABSTRACT. *Data freshness has been identified as one of the most important data quality factors in information systems. This importance increases particularly in the context of systems composed of a large set of autonomous data sources, where integrating data having different freshness may lead to semantic problems. There are various definitions of data freshness in the literature as well as different metrics to measure them, depending on the applications where they are used. This paper presents an analysis of these definitions and metrics and proposes a taxonomy to compare them. This taxonomy is useful for (i) analyzing the way data freshness is defined and used in several types of systems and (ii) studying the properties of the system that influence data freshness. We model the system and its properties as a workflow and we present a mechanism for evaluating data freshness based on the workflow representation.*

RÉSUMÉ. *La fraîcheur des données est l'un des facteurs de qualité les plus importants dans les systèmes d'information. Cette importance est accrue dans le contexte des systèmes composés d'un grand nombre de sources de données autonomes, où l'intégration des données caractérisées par des fraîcheurs différentes peut mener à des problèmes sémantiques. Dans la littérature, il existe différentes définitions de la fraîcheur des données ainsi que des métriques permettant de la mesurer, et ce en fonction des applications où elles sont employées. Cet article présente une analyse de ces définitions et ces métriques et propose une taxonomie permettant de les comparer. Cette taxonomie servira à (1) analyser la manière dont la fraîcheur des données est définie et utilisée dans différents types de systèmes, et (2) étudier les propriétés du système qui influencent la fraîcheur des données. Nous représentons le système et ses propriétés comme un workflow et nous présentons notre approche pour l'évaluation de la fraîcheur basée dans cette représentation.*

KEYWORDS: *data freshness, quality evaluation, quality metrics.*

MOTS-CLÉS : *fraîcheur des données, évaluation de la qualité, métriques de la qualité.*

1. Introduction

Data freshness has been identified as one of the most important attributes of data quality for data consumers (Shin, 1996). Some surveys and empirical studies have proved that data freshness is linked to information system success (Wang et al., 1996)(Shin, 2003). Then, achieving required data freshness is a challenge for the development of a large variety of applications. Furthermore, the increasing need to access to information that is available in several data sources introduces the problem of choosing between alternative data providers and of combining data with different freshness values. This paper presents an analysis of data freshness and its various underlying metrics in a system that integrates information of several data sources.

Examples of Data Integration Systems (DIS) are Mediation systems, where data is extracted from several sources, integrated and presented to the user, commonly using the wrapper-mediator architecture. Data Warehouse systems also extract, transform and integrate information from various, possibly heterogeneous, sources and make it available for strategic analysis to the decision makers. Other examples are federations of databases, where a key characteristic is the preservation of source autonomy and Web Portals which provide access to subject-oriented data acquired and synthesized from web sources, generally caching important amounts of data.

Several freshness definitions have been proposed for different types of data integration systems. The traditional freshness definition is called *currency* (Segev et al., 1990) and describes how *stale* is data with respect to the sources. Recent proposals incorporate another notion of freshness, called *timeliness* (Wang et al., 1996), which describes how *old* is data. Then, freshness represents a family of quality factors, or a quality dimension, with different associated metrics. Each factor best suites a particular problem or type of system.

In this paper we analyze this quality dimension and we present a taxonomy of its factors and metrics based upon the nature of the data, upon the type of application and upon the synchronization policies of the underlying management system. We analyze, in terms of the taxonomy, the way freshness is defined and used in several types of systems.

Considering a freshness factor and a method to measure it at the sources, a freshness value can be calculated for the data returned to the user. To calculate this value we should combine the different source freshness values and take into account the processes that extract, transform and convey the data to the user. We model the calculation process as a workflow of calculation activities and we present an evaluation process based on the workflow representation.

The contribution of this paper is twofold. Firstly, we analyze data freshness factors and metrics within a taxonomy. Secondly, we propose a framework for data freshness evaluation.

The rest of the document is organized as follows: Section 2 discusses the different definitions and metrics of the freshness dimension and presents the

taxonomy. Section 3 presents the freshness evaluation problem, summarizing existing works and our current investigations. Section 4 proposes a framework for data freshness evaluation. Finally, section 5 concludes with our general remarks.

2. Analysis of data freshness

Intuitively, the concept of data freshness introduces the idea of how old is the data: Is it fresh enough with respect to the user expectations? Has a given data source the more recent data? Is the extracted data stale? When was data produced? Data freshness has then not a unique definition in the literature. There are various definitions concerning different concepts and metrics, which are mainly due to the different objectives of the systems where they are used.

In this section we analyze data freshness definitions and metrics, we discuss the dimensions that impact the evaluation and enforcement of data freshness and we present a taxonomy that summarizes the discussion.

2.1. *Freshness definitions and measurement*

Data freshness comprises a family of quality factors each one representing some freshness aspect and having its own metrics. For that reason freshness is commonly mentioned as a *quality dimension* (Jarke et al., 1999). From a user point of view, we distinguish two sub-dimensions of this quality dimension:

– *Currency factor* (Segev et al., 1990): It captures the gap between the extraction of data from the sources and its delivery to the users. For example, currency indicates how stale is the account balance presented to the user with respect to the real balance at the bank.

– *Timeliness factor* (Wang et al, 1996): It captures how often data changes or how often new data is created in a source. For example, timeliness indicates how often the product prices change in a store or how often new books are added to a library.

Factor	**Metric**	**Definition**
Currency	Currency	The time elapsed since data was extracted from the source (The difference between query time and extraction time) (Segev et al., 1990)
	Obsolescence	The number of updates transactions/ operations to a source since data extraction time (Gal, 1999)
	Freshness rate	The percentage of tuples in the view that are up-to-date (have not been updated since extraction time) (Cho et al., 2000).
Timeliness	Timeliness	The time elapsed from the last update to a source (the difference between query and last update times) (Naumann et al., 1999).

Table 1. *Summary of freshness factors and metrics*

A metric is a specific instrument that can be used to measure a given quality factor. There might be several metrics for the same quality factor. Table 1 describes the metrics proposed in the literature for measuring data freshness, classified by factor. A larger description of each one can be found in (Bouzeghoub et al., 2004).

2.2. *Dimensions for freshness analysis*

In this section we analyze some dimensions that impact the analysis and enforcement of data freshness. We analyze the nature of data, the type of application and the synchronization policies of the system.

2.2.1. *Nature of data*

According to its change frequency, we classify source data in three categories:

– *Stable data*: It is data that is improbable to change. Examples are scientific publications; although new publications can be added to the source, older publications remain unchanged. Other examples are person names, postal codes and country names.

– *Long-term-changing data*: It is data that has a very low change frequency. Examples are the addresses of employees, country currencies and hotel price lists in a tourist center. The concept of "low frequency" is domain dependent; in an e-commerce application, if the stock of a product changes once a week it is considered to be low-frequency change while a cinema that changes its playbills weekly has a high-frequency change for spectators.

– *Frequently-changing data*: It is data that has intensive change, such as real-time traffic information, temperature sensor measures and sales quantities. The changes can occur with a defined frequency or they can be random. For example, restaurant menus which are updated every morning have a defined change frequency, but the account balances which are updated with every account movement have not got a defined frequency.

When working with frequently changing data, it is interesting to measure how long data can remain unchanged and minimize the delivery of expired data. However, when working with stable or long-term changing data, these questions have no sense since data does not change very often. It is more interesting to measure how often new data is created or how old is the data.

2.2.2. *System features*

The freshness of the data returned to the user depends on the freshness of extracted data but also on the processes that extract, integrate and deliver this data. These processes are very important because they can introduce additional delays. We distinguish three main families of architectural techniques: those that calculate data when a new query is posted, those that cache the data most frequently used, and those that materialize the data needed to answer user queries. When using

materialization, data is stored for some time in the data integration system repositories, which decreases its freshness. The features of these three categories of systems are summarized below:

– *Virtual systems*: The system does not materialize any data so all queries are calculated when they are posed. The users send their queries to the system and wait the response. The system queries the relevant sources and merges their answers in a global answer that is delivered to the user. Examples are pure virtual mediation systems and query systems in database federations.

– *Caching systems*: The system caches some information, typically some source relations that are frequently accessed or the result of some frequent queries. The system estimates the time during which the data will be up to date and invalidates it when passed this time. The users pose their queries to the system and if the information required to answer the queries is stored in the cache, the system delivers it to the user. If the information is not stored in the cache or it is invalidated, the system queries the sources as in virtual systems and possibly refreshes cache data. Examples are caching and replication systems.

– *Materialized systems*: The system materializes large volumes of data which is refreshed with respect to a certain scenario. The users pose their queries and the system answers them almost using the materialized data. Examples are data warehousing systems and web portals that support materialization.

Virtual systems are conceived to retrieve data as current as possible, returning a current state of the source data. In caching systems, some level of staleness is allowed but the gap between source states and the integration system state should be relatively small. In materialized systems that gap can be greater, but its magnitude depends on the concrete applications.

2.2.3. *Synchronization policies*

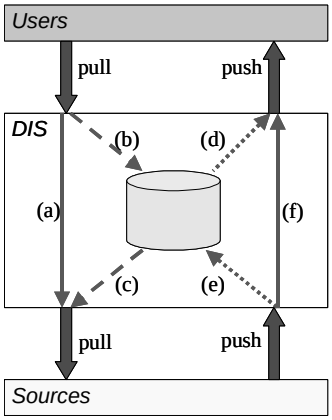
The way the data integration system (DIS) is implemented influences the freshness of the data delivered to the users. Particularly, the synchronization between the sources, the DIS and the users has impact in data freshness because it introduces delays. For example, a system that synchronizes updates each end of the day may provide data which is not fresh enough for the expectations of a given user.

According to the interaction between the DIS and the sources, the extraction processes can have *pull* or *push* policies. With pull policy, the DIS queries the sources to obtain data and with push policy, the source sends data to the DIS. The notification of new available data can come from an active source (e.g. using triggers) or can be determined by the DIS continuously polling the source. Active sources can have their own policies as sending each updated tuple, or sending sets of tuples every regular periods of time or when changes surpass a threshold. Pull policies can also be driven by temporal or non-temporal events.

According to the interaction between the DIS and the users, the query processes can also have pull or push policies. With pull policy, users directly pose queries to

the DIS. With push policy, users subscribe to certain queries and the DIS regularly conveys response data to the users. Push policies can also be driven by temporal or non-temporal events.

Combining the previous interactions between users, DIS and data sources leads to six possible configurations which are shown in figure 1. With synchronous policies, the user directly accesses source data. With asynchronous policies, the DIS answers user queries using materialized data and asynchronously, the materialized data is refreshed from source data.



Configurations are named with the user-DIS policy followed by the DIS-source policy, (/) represent asynchronisms and (-) synchronisms.

Synchronous configurations:

- pull-pull: arrow (a)
- push-push: arrow (f)

Asynchronous configurations:

- pull / pull: arrows (b) and (c)
- pull / push: arrows (b) and (e)
- push / push: arrows (d) and (e)
- push / pull: arrows (d) and (c)

Figure 1. *Synchronization policies*

Asynchronous policies introduce delays. The refresh frequency of the DIS repository is important to evaluate the freshness of retrieved data. When pushing data to the user, the push frequency is also important.

In systems where there are heterogeneous data sources with different access constraints and users with different freshness expectations, it is important to support and combine several kinds of policies.

2.3. *A taxonomy of freshness works*

In this section we present a taxonomy that summarizes this discussion and allows comparing different proposals for freshness evaluation. The taxonomy is composed of the previously described dimensions: (i) nature of data, (ii) application type and (iii) synchronization policies.

Nature of data is a user-oriented dimension which qualifies the properties of source data from a user point of view. But not all the combinations of nature of data and freshness factors are interesting. On the one hand, when data changes very frequently, there is not interest in measuring timelines, which captures stable data

behaviour. On the other hand, there is no sense to evaluate the currency of long-term and stable data because they are almost always current as they do not change very often. In the latter case, the system can assure currency even without explicit evaluation. The other combinations need evaluation to determine the freshness level. For them, the development of evaluation tools is interesting. Table 2 shows the relation, indicating when data freshness can be assured without evaluation and when it is interesting to evaluate it (V) or not (X).

	Frequently changing	**Long-term changing**	**Stable**
Timeliness	X	V	V
Currency	V	assured	assured

Table 2. *Interesting combinations of freshness factors and nature of data*

System features and *synchronization policies* are system-oriented dimensions which describe the system relation with data freshness. Not all the combinations between system features and synchronization policies are valid. Table 3 shows the interrelations between them, indicating the valid combinations.Virtual systems only support the pull-pull configuration. Materialized systems support the configurations having an internal repository to store materialized data. Caching systems support the configurations that pull source data (synchronous and asynchronous). The push-push configuration is not usually implemented in the three system types discussed.

	pull-pull	**pull/ pull**	**pull/ push**	**push/ pull**	**push/ push**
Materialized	X	V	V	V	V
Caching	V	V	X	V	X
Virtual	V	X	X	X	X

Table 3. *Valid combinations of system features and synchronization policies*

The user-oriented and the system-oriented dimensions are orthogonal. Virtual, caching or materialized systems (with their valid combinations of synchronization policies) can be built to query different types of data.

The system-oriented dimensions are also orthogonal to the freshness factors, because user interest in freshness is independent to the way the system is implemented. However, the metrics for the currency factor are related to the system implementation. In virtual systems the main interest is the response time, so the currency metric is appropriate. In caching systems all the metrics have been identified as interesting, as different existent applications evaluate them. In materialized systems, currency and obsolescence have been used. Table 4 shows the co-relation of all the taxonomy dimensions:

	Frequently changing	Long-term changing	Stable
Virtual	Currency	Timeliness	Timeliness
Pull-pull			
Caching	Currency Obsolescence Freshness-rate	Timeliness	Timeliness
Pull-pull **Pull/pull** **Pull/push**			
Materialized	Currency Obsolescence	Timeliness	Timeliness
Pull/pull **Pull/push** **Push/pull** **Push/push**			

Table 4. *Co-relation of all the taxonomy dimensions*

The technical problems to solve for each combination are quite different. For example, enforcing currency in a materialized system implies developing efficient update propagation algorithms to deal with consistency problems, while evaluating timeliness in virtual systems is quite independent on the query rewriting algorithms and is dominated by source data timeliness. In next section we discuss the freshness evaluation problems.

3. The data freshness evaluation problems

In this section we analyze several types of systems that evaluate data freshness in terms of the taxonomy. A larger description of the state of the art and research problems is presented in (Bouzeghoub et al., 2004). We finally enounce our current investigations in the line of data freshness evaluation, which are detailed in section 4.

3.1. Some systems that consider data freshness

In this section we analyze several types of systems that evaluate data freshness and we describe the goals and problems that they present. Table 5 summarizes the proposals in terms of the taxonomy presented before.

3.1.1. Data warehousing systems

In data warehousing systems, freshness is studied through the *currency factor* in the context of view materialization. Traditional query optimization algorithms are extended to take into account the materialized views. In (Gal, 1999), a cost model has been proposed for balancing the query generation and data transmission cost on the one hand, and the *obsolescence* cost on the other hand.

Materialization introduces potential inconsistencies with the sources and warehouse data may become out-of-date (Zhuge et al., 1997). The view maintenance problem consists in updating a materialized view in response to changes arisen at source data. Most of the work concentrates in assuring DW consistency for different types of views and refresh strategies (Gupta et al., 1995). A key problem in the last years has been the selection of a set of views to materialize in order to optimize the query evaluation and/or the maintenance cost, possibly in the presence of some constraints (Gupta, 1997)(Yang et al, 1997). Data freshness is implicitly considered when defining the update propagation processes. In most works, the update propagation processes are triggered by sources when the amount of changes is greater than a threshold or are executed periodically (pull/push and pull/pull policies). In (Theodoratos et al., 1999), they propose an algorithm that takes as input the user expectations for data currency and determines the minimal update frequencies that allow achieving these values (pull/push policy).

3.1.2. Mediation systems

In classical mediation systems, freshness is also studied through the *currency factor*. In (Hull et al., 1996), authors propose the construction of *Squirrel mediators*, which combine virtual and materialized data, and formally proof that the freshness of the returned data is bounded. They combine pull-pull and pull/pull policies.

New proposals take into account the *timeliness factor*. It is used as a quality metric to compare among sources and to filter the data returned to the user. In (Naumann et al., 1999), they study how to propagate a set of quality factors from several heterogeneous sources to the mediator. They propose a virtual scenario with pull-pull policy.

3.1.3. Caching systems

In caching systems, data is considered fresh when it is identical to data in the sources, so freshness is represented by the *currency factor* and measured with its metrics (currency, obsolescence, freshness-rate). An important problem is keeping cache data up-to-date. Traditional cache proposals estimate the *time-to-live* (TTL) of an object (Bright et al., 2002), so the cache can store frequently changing data as well as long-term changing data. When the TTL has expired the object is invalidated in the cache, so in the next access the object will be directly read from the source (pull-pull and pull/pull policies). In (Cho et al., 2000), they study synchronization policies for cache refreshment and experimentally verify their behavior. They measure freshness with two metrics: currency (called age in the paper) and freshness-rate. In (Li et al., 2003), they balance response time and invalidation cycles for assuring data currency. In (Bright et al., 2002), they propose the use of *latency-recency* profiles to adapt caching algorithms to user currency requirements.

Newer proposals combine caching and materialization techniques. In (Labrinidis et al., 2003), they propose an algorithm to select which WebViews (html fragments derived from a database) to materialize without exceeding a given currency threshold.

3.1.4. Replication systems

In a replication context, data at a slave node is totally fresh if it has the same value as the same data at the master node, i.e. all the refresh transactions for that data have been propagated to the slave node (Gancarski et al., 2003). Freshness is studied by means of the *currency factor* for frequently changing data.

In (Gancarski et al., 2003), they determine the minimum set of refresh transactions needed to guarantee that a replica is fresh enough with respect to the user freshness requirements for a given query. They propose a load-balancing algorithm that takes freshness into account to decide when to refresh a replica. They follow pull-pull and pull/pull policies.

Works	**Measurement**	**Nature of data**	**Application type**	**Synchroniz Policy**
Materialization for query processing	Obsolescence	Frequently changing	Virtual, materialized	Pull-pull, pull/pull
View maintenance	Currency	Frequently changing	Materialized	Pull/pull, pull/push
View maintenance policies	Currency	Not specified	Materialized	Pull/pull
Selection of views to materialize	Currency	Frequently changing	Materialized	Pull/pull, pull/push
Mediation design (with materialization)	Currency	Not specified	Virtual, materialized	Pull-pull, pull/pull
Source selection in mediation	Timeliness	Not specified	Virtual	Pull-pull
Cache refreshment	Currency, obsolescence, freshness-rate	Frequently changing / long-term changing	Caching	Pull-pull, pull/pull, Pull/push
Cache refreshment (with materialization)	Freshness-rate	Frequently changing	Caching, materialized	Pull-pull, pull/push
Replica refreshment	Currency, obsolescence	Frequently changing	Caching	Pull-pull, pull/pull

Table 5. *Summary of proposals*

3.3. Our current investigations

The analysis of existing works in terms of the taxonomy (Bouzeghoub et al., 2004) suggested open problems in the specification of user expectations, the acquisition of source freshness measures and the formulation of cost models for the query evaluation and update propagation processes of heterogeneous systems. This knowledge can be used both for developing auditing tools that estimate the freshness of an existing system and for designing a system driven by freshness expectations.

Among these problems, we are interested in the development of an auditing tool for evaluating data quality, in particular data freshness, and deciding if user quality expectations can be achieved. The auditing tool should take as input some metadata describing the data integration system, the sources and the query classes, as well as measures of the actual quality of source data and user quality requirements. The tool should return a measure of the quality of the data returned to the user.

The tool can be used to verify if the data returned to the user fulfills its freshness requirements and if not, the evaluation can aid for determining which parts of the system should be improved. Furthermore, the tool can be used to compare alternative processes which can implement the data integration system either accessing to different sources or computing the resulting information following different algorithms. The comparison can: (i) determine which processes achieve freshness expectations, (ii) order the candidate processes by their freshness, or (iii) propose the user only the most adequate process (or processes). Next section proposes a framework to develop such tool.

4. A framework for data freshness evaluation

In this section we present a framework for data freshness evaluation. We represent the DIS integration process as a workflow of possibly autonomous calculation activities and we analyze how to measure data freshness within the workflow. We also discuss how to compare freshness values with user freshness expectations and how to use the comparison to enforce freshness.

4.1. A graph representation of a DIS

The DIS integration system can be viewed as a workflow where different calculation activities perform the different tasks that extract, transform and convey data to end-users. Each activity takes input data from sources or other activities and produces result data that can be used as input for other activities. Then, data traverses a path from sources to user queries where it is transformed and processed according to the system logics. The data produced by an activity can be immediately consumed by other activities or it can be materialized for being queried later. Note that this notion of activity can represent processes of different complexities, from simple SQL operations to complex transformation procedures that can execute autonomously.

The workflow representation is general enough to represent different types of systems. We mention some of them:

– In classical mediation systems the workflow activities are composed of wrappers and mediators.

– In DW systems, the refreshment processes can also be seen as a workflow of activities (Bouzeghoub et al., 1999). The activities can be complex processes that

produce the views of the different DW repositories, i.e. the operational data store, the corporate DW and the data marts.

– In caching systems, the refreshment processes can be seen as simple workflow processes which store data in the cache and ask the web for additional data.

Figure 2 sketches the workflow representation. On the bottom diagram there are remote source relations (R_i). On the middle diagram there are the different activities (A_i) whose inputs are source data. The arrows indicate that the output node uses the data returned by the input node. The activities that directly take input data from source relations are the *wrappers* that perform the data extraction from sources. The other activities take input data, direct or indirectly, from wrappers. On the top diagram there are the user query classes (Q_i) representing families of queries that can be executed using the data produced by activities.

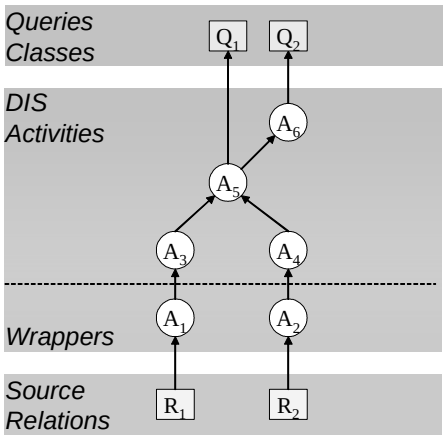


Figure 2. *A workflow representation of a DIS*

4.1.1. The calculation dag

Formally, we represent the DIS workflow by means of a directed acyclic graph (dag) that describes the involved activities, their inputs and outputs. The dag shows the data paths from source relations to user queries within the different activities.

Definition 1. A *calculation dag* is a dag G defined as follows: The nodes of G are of three types: *source nodes* (with no input edges) that represent the source relations, *target nodes* (with no output edges) that represent query classes and *activity nodes* (with both input and output edges) that represent the different activities that calculate the set of target nodes from the source nodes. The edges of G represent that a node is calculated from another (the data flows in the sense of the arrow). □

Example 1. Consider the calculation dag of figure 2 representing the refreshment process of a simple DW system that integrates information of source relations R_1 and

R_2. Activities A_1 and A_2 are the wrappers, activities A_3 and A_4 perform the cleaning and activity A_5 integrates and materialize data to support user queries of class Q_1. Activity A_6 calculates a monthly summary and materializes it for other user queries (Q_2). The edges entering to A_5, for example, indicate that data produced by A_3 and A_4 is used as input in the integration process and the edges coming out A_5 indicate that integrated data is used as input for A_6 and Q_1. □

4.2. Labeling the calculation dag

The freshness of the data delivered to the user depends on the execution delay of the system, that is the length of time from data extraction to data delivery. This length of time is influenced not only for the execution cost of each activity (the time the activity needs for executing) but also for the delays that can exist between the executions of consecutive activities. These delays are determined by the execution policies of the system and the access constraint of the sources.

In this section we define some properties that describe the execution delay of the system. We will associate them as labels of the calculation dag.

4.2.1. Some properties

We analyze the properties that impact in the execution behavior of the system, namely the execution policies, synchronization delays, access constraints and execution costs.

Execution policy: We consider three possible execution policies for an activity depending on the synchronization with predecessor and successor nodes:

– *Input-synchronous*: The activity is synchronized with its predecessor nodes. The activity execution starts when it receives a notification about new available data in one of its inputs. For wrapper activities, the notification comes from the source. For the other activities, the notification comes from a predecessor node announcing that it has finished its execution and has produced new data.

– *Output-synchronous*: The activity is synchronized with its successor nodes. The activity execution starts when it receives a notification of some successor that needs new data, then, the activity has to execute to provide this new data. For the activities that directly answer user queries, the notification comes from the processes that manage user queries. For the other activities, the notification comes from a successor node announcing that it has to execute and needs new data as input.

– *Asynchronous*: The activity is not synchronized with predecessor nor successor nodes. The activity execution starts when the activity receives a notification from the system. We consider that the system sends periodic notifications, then there is a fixed amount of time (period) between two consecutives executions.

In systems with homogeneous synchronization policies (see section 2.2.3) the execution policies of its activities follow well-known patterns. For example, in a pull-pull system all activities have output-synchronous policies and in a pull/pull

system, the activities that answer user queries from materialized data follow output-synchronous policies while the activities that refresh the materialized data follow asynchronous policies. In heterogeneous systems that combine several synchronization policies the synchronization between every pair of activities should be studied.

Synchronization delay: When two consecutive activities in a path have different execution frequencies (due to different execution policies), the data produced by the former must be materialized for being queried later by the latter. In this case, there is a synchronization delay. The synchronization delay between two activities is the amount of time passed between the end of the execution of one activity and the start of the other. The synchronization delays are very important in the evaluation of data freshness because they introduce extra waiting time for data and consequently they decrease the data freshness.

If two consecutive activities are synchronized and the second activity executes always a fixed amount of time after the first one, the synchronization delay is well-known. If activities are not synchronized, the synchronization delay has to be estimated in the worst case. The algorithms to determine the delays between two activities (or estimate them in the worst case) are presented in (Peralta et al., 2004).

Example 2. Consider the evaluation of timeliness in the calculation dag of previous example and suppose that wrappers (A_1 and A_2) execute weekly following asynchronous policy, cleaning and integration processes (A_3, A_4 and A_5) execute after the data extraction (input-synchronous policy) and the monthly summary (A_6) executes at the end of each month (asynchronous policy). Imagine that source R_2 materialize data, which is read by A_2 two days later (delay is 2 days) and source R_1 does not materialize data (no delay). As A_3, A_4 and A_5 are input-synchronous there is not delay with their predecessors. However, as A_5 and A_6 have different policies with different periods (7 and 30 days) the data produced by A_5 can have been materialized for almost a week when read by A_6, then the delay will be a week in the worst case. The delays with target nodes are the refresh periods of the materialized data, in the worst case. Figure 3a shows the calculation dag labeled with synchronization delays. □

Access constraint: In some situations, the sources do not allow the system to continuously query them, or it is very expensive to query a source very often. We can express such constraints giving a maximum access period for the data extraction. The *access period* is the lowest time interval that a source allows between two consecutive data extractions. Note that in the presence of access constraints, the wrapper must periodically materialize data to assure the availability of source data (asynchronous execution policy). The access constraint should be explicitly defined by the source provider, the system designer or both.

Processing cost: The processing cost of an activity is the amount of time, in the worst case, necessary for reading input data, executing and building result data. Each activity needs some time for executing and obtaining a result; this amount of time is the processing cost. There are several delays due to the execution of the activity. For

wrappers, it involves the time necessary for communicating with the source (sending the request and waiting for the response), the time for computing the extraction query and the time for materializing the changes (if needed). For the other activities, it involves the time for reading input data, computing and the time for materializing the result (if needed). Communication time can be estimated using statistics of previous executions. Computation and materialization times can be estimated using cost models. Our approach is independent of the cost model used but the estimation depends on it. The determination of the appropriate cost model depends on the freshness metric and on the three dimensions of the taxonomy.

Example 3. Consider the evaluation of timeliness in the calculation dag of previous example. In such a system, the communication costs and the update propagation costs are negligible compared to synchronization delays (week, month). A cost model for this system can neglect the execution costs of the nodes. However, if the cleaning activity A_4 needs user interaction and the cleaning process can have a duration of two days, this cost should be modeled. □

4.2.2. Labeling the calculation dag

We label each node of the calculation dag with a descriptive name. In addition, we label activity nodes with the processing cost, execution policy and access constraint (if there is one). We label each edge with the synchronization delay.

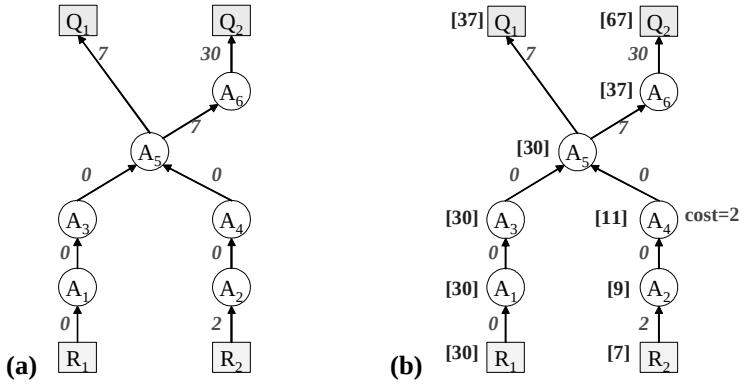


Figure 3. (a) *Synchronization delays,* **(b)** *Freshness propagation*

4.3. Evaluating freshness within the graph

In this section we specify how to propagate freshness values within the graph, calculating the freshness of the intermediate data produced by each node. We firstly give an intuitive idea of the freshness calculation method and we present a recursive definition. We also enunciate some lemmas that show another way to calculate freshness in terms of the dag paths. Such calculation allows the definition of some strategies to enforce freshness.

Intuitively, the freshness of the data produced by a node depends on the freshness of data at the moment of reading it (the freshness of data produced by the predecessor node plus the synchronization delay) and the time the node needs for executing (the execution cost). To calculate the freshness of a node we add such values. When the node reads data from several input nodes, input freshness values should be combined. As we are interested in an upper bound of freshness we take the worst case (the maximum). We recursively define the freshness of a node as follows:

Definition 2. The *freshness* of an activity or target node in a calculation dag G is the maximum sum of the freshness of a predecessor node, plus the synchronization delay between nodes, plus the processing cost of the node. The *freshness* of a source node is its actual freshness.

– For a source node A: Freshness(A) = ActualFreshness(A)

– For an activity or target node A: Freshness(A) = max {Freshness(B) + delay(B,A,G) /B ∈ predecessor(A,G)} + cost(A,G)

The *ActualFreshness* is a measure of the freshness of data in a source. The *cost* and *delay* functions return the corresponding properties and the *predecessor* function returns the predecessors of a node in the dag. □

Example 4. Figure 3b continues previous example, adding the freshness of each node between square brackets. We suppose that the actual freshness of sources is 30 and 7 days respectively. Freshness of activities and queries is calculated using the previous definition (remember that most activity costs have been neglected, except A_4 which cost is 2). For A_5 values of both inputs are compared, 30 and 11 respectively, taking the maximum. □

Intuitively, if we consider a sequence of activities that execute one after the other (a path in the calculation dag), the freshness of data after executing the last activity is the sum of the freshness in the moment data was read by the first activity (initial freshness), plus the cost of the path, i.e. the time to execute all the activities in the path (execution cost) plus the time activities wait for start execution (synchronization delay). We define the path cost as follows:

Definition 3. Given a path $[A_0,A_1,\dots A_p]$ in the calculation dag G from a source node A_0, we define the *path cost* as the sum of processing costs (of nodes A_1 to A_p) plus the synchronization delays between all the nodes:

$$\text{PathCost}([A_0,\dots A_p]) = \Sigma_{x=1..p}(\text{cost}(A_x,G)) + \Sigma_{x=1..p}(\text{delay}(A_{x-1},A_x,G))$$ □

Observe that for each node, there is a path for which we add all synchronization delays and processing costs to the source actual freshness and we obtain the freshness of the node.

Exemple 5. In the previous example, the freshness of A_6 can be calculated adding the freshness of source R_2 (7), plus the delays (2,0,0,7) and the costs (0,2,0,0) in the path $[R_2, A_2, A_4, A_5, A_6]$. □

Then, the freshness of a node can be calculated as the cost of the most expensive path from a source relation plus the source actual freshness. The following lemmas specify it:

Lemma 1. For a given node A_p, there exists a path $[A_0,A_1,\dots A_p]$ from a source node A_0 that verifies:

$$Freshness(A_p) = Freshness(A_0) + PathCost([A_0,A_1,\dots A_p])$$ □

Lemma 2. For a given activity node A_p, freshness is given by the cost of the most expensive path:

$$Freshness(A_p) = \max \{Freshness(A_0) + PathCost([A_0,A_1,\dots A_p]) / [A_0,A_1,\dots A_p] \text{ is a path from a source node}\}$$ □

Lemma 1 states that there exists a path in the dag that determines the freshness of each node and lemma 2 states that this path is the most expensive one (the critical path). Demonstration can be found in (Peralta et al., 2004). The existence of a critical path allows the use of a large spectrum of algorithms for optimizing a workflow of activities. Next section discusses its use when user requirements are not achieved.

4.4. Enforcing freshness

Our goal is to provide at the query level the data freshness expected by the users. To know if user freshness expectations can be achieved by the system, we can propagate freshness values (as defined in definition 2) and compare them with those expected for queries. If the propagated freshness values are lower than user expected values then freshness can be guaranteed. If the propagated freshness values are greater than user expected values, we have to improve the system design to enforce freshness or negotiate with source data providers or users to relax constraints.

When freshness cannot be assured for a user query, there exists at least one path from a source relation where propagated freshness is higher than expected freshness. There are several alternatives to enforce freshness in a path:

– Negotiating with users to relax freshness expectations. It should be followed when users expectations are too high for the data that can be effectively obtained from sources.

– Negotiating with source data providers to relax source constraints. Sometimes the system hardware can be powered to support more frequent accesses to the sources. Other times, this alternative implies demanding and eventually paying for a better service, for example, receiving data with a lower actual freshness.

– Improving the design of the activities of the path in order to reduce their execution cost. This implies the design of the system to reduce the execution cost of the activities. Sometimes the changes can be concentrated in the critical path that slows the system. Other times a complete reengineering of the whole system is necessary, either changing the implementation of the activities, the synchronization policies, the decisions of which data to materialize or even the hardware.

– Synchronizing the activities of the path in order to reduce the delay between them. This implies finding the most appropriate execution frequencies for some activities respecting possible source access constraints. The main difficulty resides in the synchronization of activities having several inputs, sometimes with different policies and refresh frequencies.

4.5. Illustrating example

We summarize the proposal with an example. Consider a data integration system built for retrieving meteorological information, which is illustrated in figure 4. Freshness is evaluated with the timeliness factor.

There are three source relations: R_1 with real time satellite meteorological predictions, R_2 which is a dissemination database updated once a day, and R_3 with information about climatic sensors which is published with a three hours delay. Source actual freshness is 0 (negligible), 24 and 3 hours respectively.

The goal of the system is to provide fresh meteorological information to solve four types of queries: Q_1 (historical information about climate alerts), Q_2 (detailed data comparing predictions), Q_3 (aggregated data about predictions) and Q_4 (aggregate data about climate measurements). Users expect that the freshness of retrieved data does not exceed 168, 72, 48 and 2 hours respectively.

The DIS is composed of nine activities that process the information performing the data extraction, filtering, integration and aggregation. Figure 4 shows the relationship between activities and their costs expressed in hours (inside each node,

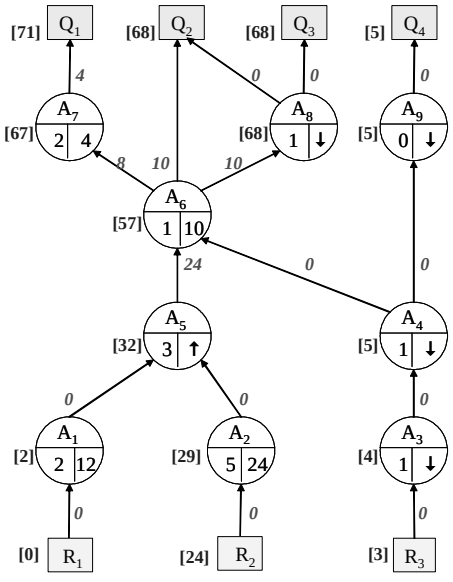


Figure 4. *Calculating freshness in a meteorological system*

at the left). Activities execute when queries are posed (output-synchronous: ↓), when new data is extracted (input-synchronous: ↑) or asynchronously (execution period is in indicated inside the nodes, at the right). The delays are shown near the edges.

Freshness values are shown between square brackets near each node. They are calculated following definition 2, adding execution cost and synchronization delay to the freshness of predecessors.

Data for solving Q_1 has a freshness of 71 hours which satisfy user expectations of 168 hours. This means that the system can be relaxed and activities in the paths from sources to Q_1 can be executed less frequently. Data for solving Q_3 does not achieve user expectations (68 versus 48 hours). Analyzing the critical path to Q_3 ([R_2, A_2, A_5, A_6, A_8, Q_3]) some activities can be synchronized to reduce the delays and attain freshness expectations (for example executing A_6 immediately after A_5). The execution cost of some activities can be reduced too (for example replacing wrapper A_2 for a more performing one). However, even neglecting the cost of the activities in the paths to Q_4, freshness expectations cannot be achieved because the actual freshness of R_3 is too high. The solution should be a negotiation with users and/or source providers.

5. Conclusion

Data freshness represents a family of quality factors and metrics. In this paper we have analyzed these factors and metrics and the features that influence the data freshness evaluation, namely the type of application, the synchronization policy and the nature of data. We presented a taxonomy of freshness factors and metrics and we used it to classify existing works.

We have also presented our current investigations in the line of developing an auditing tool for data freshness evaluation. We proposed a framework for performing such evaluation, which models the DIS integration process and its properties in terms of a labeled calculation dag. We discussed data freshness evaluation and enforcement solutions as graph traversal mechanisms.

Data freshness is a *first class* quality dimension which is more and more required by end-users. Solving the problems of evaluating and enforcing data quality opens a door to consider data production as any other item production.

6. References

Bouzeghoub M., Fabret F., Matulovic-Broqué M., « Modeling Data Warehouse Refreshment Process as a Workflow Application », in proc. of the *Int. Workshop on Design and Management of Data Warehouses DMDW'99*, Germany, 1999.

Bouzeghoub M., Peralta V., « A Framework for Analysis of Data Freshness », in proc. of the *Int. Workshop on Information Quality in Information Systems IQIS'2004*, France, 2004.

Bright L., Raschid L., « Using Latency-Recency Profiles for Data Delivery on the Web », in proc. of the *28th Int. Conf. on Very Large Databases VLDB'02*, China, 2002.

Cho J., Garcia-Molina H., « Synchronizing a database to improve freshness », in proc. of the *2000 ACM Int. Conf. on Management of Data SIGMOD'00*, USA, 2000.

Gal A., « Obsolescent materialized views in query processing of enterprise information systems », in proc. of the *1999 ACM Int. Conf. on Information and Knowledge Management CIKM'99*, USA, 1999.

Gancarski S., Le Pape C., Valduriez P., « Relaxing Freshness to Improve Load Balancing in a Cluster of Autonomous Replicated Databases », in proc. of the *5th Workshop on Distributed Data and Structures WDAS*, Greece, 2003.

Gupta A., Mumick I., « Maintenance of Materialized Views: Problems, Techniques, and Applications », *Data Engineering Bulletin*, June 1995.

Gupta H., « Selection of Views to Materialize in a Data Warehouse », in proc. of the *6th Int. Conf. on Database Theory ICDT'97*, Greece, 1997.

Hull R., Zhou G., « A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches », in proc. of the *1996 ACM Int. Conf. on Management of Data SIGMOD'96*, Canada, 1996.

Jarke M., Jeusfeld M., Quix C., Vassiliadis P., « Architecture and Quality in Data Warehouses: An Extended Repository Approach », *Info Systems*, vol.24(3): 229-253,1999

Labrinidis A., Roussopoulos N., « Balancing Performance and Data Freshness in Web Database Servers », in proc. of the *29th Int. Conf. on Very Large Data Bases VLDB'03*, Germany, 2003.

Li W.S., Po O., Hsiung W.P., Selçuk Candan K., Agrawal D., « Freshness-driven adaptive caching for dynamic content Web sites », *Data & Knowledge Engineering DKE*, vol. 47(2): 269-296, 2003.

Naumann F., Leser U., « Quality-driven Integration of Heterogeneous Information Systems », in proc. of the *25th Int. Conf. on Very Large Databases VLDB'99*, Scotland, 1999.

Peralta V., Bouzeghoub M., Evaluating Data Freshness in Data Integration Systems, technical report, Université de Versailles, France, 2004.

Segev A., Weiping F., « Currency-Based Updates to Distributed Materialized Views », in proc. of the *6th Int. Conf. on Data Engineering ICDE'90*, USA, 1990.

Shin B., « An exploratory Investigation of System Success Factors in Data Warehousing », *Journal of the Association for Information Systems*, vol. 4(2003), 141-170, 2003.

Theodoratos D., Bouzeghoub M., « Data Currency Quality Factors in Data Warehouse Design », in proc. of the *Int. Workshop on Design and Management of Data Warehouses DMDW'99*, Germany, 1999.

Wang R., Strong D., « Beyond accuracy: What data quality means to data consumers », *Journal on Management of Information Systems*, vol. 12, 4:5-34, 1996.

Yang J., Karlapalem K., Li Q., « Algorithms for materialized view design in data warehousing environment », in proc. of the *23rd Int. Conference on Very Large DataBases VLDB'1997*, Greece, 1997.

Zhuge Y., Garcia-Molina H., Wiener J., « Multiple View Consistency for Data Warehousing », in proc. of the *13th Int. Conf. on Data Engineering ICDE'97*, UK,1997.