Non-Intrusive Assessment of Organisational Data Quality

Binling Jin and Suzanne M. Embury
Department of Computer Science,
University of Manchester,
Oxford Road, Manchester, M13 9PL, U.K.
{BJin|SEmbury}@cs.man.ac.uk

Abstract: Many organisations are becoming increasingly aware that the usefulness of their data is limited by its poor quality. Surprising (and sometimes alarming) proportions of data in databases are inaccurate, incomplete, inconsistent or out of date. One-off data cleaning methods can help the situation in the short term, but they are costly and do little to improve data quality in the long term. However, in order to plan and monitor the progress of long term data quality improvement programmes, it is necessary to be able to assess the quality of data across an organisation. Since resources for such programmes are generally limited, and since much of the data in question resides in mission critical systems, it is vital that these assessment activities do not intrude on normal day-to-day business processing.

In this paper, we present an approach to assess organisational data quality on a regular basis, which does not delay or disrupt revenue-generating data processing activities. We have adapted techniques from distributed query processing and distributed integrity checking to produce a system that takes account of the workload at each local site when distributing the defect checking work in the distributed information system. The approach assesses the data quality during the periods of low system activity, and ships data defects found to a global site, which time stamps them and records them for later analysis.

1 Introduction

In recent years, many organisations have come to regard their accumulated data as a valuable asset that can be exploited in creative ways, to improve revenues and customer satisfaction. However, this increased use of data has also led to the discovery that much of it is of very poor quality [Redman1996]. Data is often found to be inaccurate, inconsistent, incomplete and out of date. One-off data cleaning efforts can help in the short term, but they are expensive and do nothing to improve data quality in the long term [English1999]. Ideally, organisations need to discover the causes of poor data quality and institute improvement programmes to remove them.

However, a fundamental element of any improvement program is the ability to assess the initial state of whatever we wish to improve. In order to plan an effective improvement programme, and make best use of the scarce resources available for it, an organisation must have a clear picture of the current quality of data in its systems. This allows the improvement effort to be directed at areas of greatest potential benefit, and provides a baseline against which the success of the programme can be gauged. In addition, it is necessary to reassess quality levels continuously at various points throughout the improvement programme, in order to track its effects and the degree of improvement achieved. What is required, therefore, is some means of regularly assessing the quality of an organisation's data at an acceptable cost, both in terms of staff resources and processing time. Most importantly, since much of the data to be assessed will be stored in mission critical systems, it is necessary that normal (revenue generating) business processing is not disrupted or delayed by this assessment activity.

Although some aspects of data quality cannot be assessed by purely automatic means, levels of other aspects can be gauged by issuing queries over data, which search for data defects of a particular type. The number of defects located, relative to the amount of data searched by the queries, can give an indication of how far quality levels differ from the targets set by the organisation. For example, *consistency* and (in some cases) *accuracy* of organisational information can be assessed by issuing queries which compare data values stored in different information systems, while *completeness* can sometimes be determined by queries which identify records with certain null attributes, or where sets of records in different data sets cover different sets of real world data.

Given that certain forms of organisational data quality can be measured using cross-database queries, when are such queries to be executed? While it might be thought that they should be executed whenever an assessment of the data quality is required (i.e. at the beginning of the improvement programme, and at regular intervals throughout it) there are two disadvantages of this approach:

- Queries that assess data quality levels by comparing several data sets are typically very time consuming to evaluate. This means that execution of many such queries at one time requires significant data processing resources, which the owners of the databases involved may be unwilling to release (due to the potential delay to mission critical or revenue generation processing activities).

- Periodic assessment of data quality (e.g. every three months) provides a very imperfect and incomplete picture of the quality of data present in the system at the time of the assessment. For example, defects which entered the system since the last D.Q. assessment, but which were cleaned up before the next assessment activity, will not be detected. Also, the assessment provides only very coarse-grained details of when defects entered, or were removed from, the system (e.g. “some time in the last three months). Thus, much information vital to the correct interpretation of the root causes of data defects goes unrecorded.

An alternative strategy is to execute the queries continually (or, at least, whenever data is changed in a way that might produce a change in their results). This approach, which is essentially a more relaxed form of integrity checking [Nicolas1982], provides “perfect” information about data quality levels, at all times, but at an extremely severe cost to the organisation.

In this paper, we discuss a compromise approach in which system owners specify how much data processing resource they wish to give up to the data quality assessment activities, and at which times assessment can occur. For example, the owner of a system may be happy to allow defect detection queries to be run between the hours of midnight and 4.00am every day, when there is very little business processing activity for the system to support. We have adapted ideas from distributed query processing [Haas1997] and distributed integrity checking [Chawathe1996] to produce a system which schedules the distributed execution of data defect queries according to the workload patterns specified for each site, and which records details of any defects found, for later analysis by the data quality improvement team. This approach has the advantage that assessment activities can be run often, thus providing a more complete picture of the changing levels of data quality, without intruding on important revenue-generating processing activities. However, it has the disadvantage that it may sometimes produce an inaccurate representation of the data quality levels, due to the delays which can arise in distributed query processing. We have therefore produced a cost model, which allows the system to allocate work between the distributed systems in a way that minimises the inaccuracies in the results.

The remainder of this paper is organised as follows. Section 2 discusses existing approaches to the assessment of data quality levels in information systems. In Section 3, we give an overview of our system for non-intrusive assessment of data quality, while in Section 4 we present its shortcomings in terms of accuracy and currency of the information generated, and describe how our cost model allows us to minimise these shortcomings. Finally, Section 5 concludes and makes some suggestions for future directions for this work.

2 Existing Approaches to Data Quality Assessment

A number of approaches for assessing data quality (DQ) have been proposed. These approaches can be mainly divided into two classes: *assessment of subjective DQ* and *assessment of objective DQ*. The former measures individual stakeholders' subjective assessments of the DQ, typically using a questionnaire-based approach [Huang1999]. Analysis of the answers given to questions posed by a DQ questionnaire can help to identify specific problem areas, and to suggest actions which can be taken to monitor and improve the organisation's DQ. Subjective DQ measures are clearly valuable as a means of determining user satisfaction (or dissatisfaction) with their data. However, this method may not be suitable for measuring intrinsic data quality characteristics, as the results depend on the individuals surveyed and their intuition about what may be many years of experience working with a changing information system.

Assessment of objective DQ, on the other hand, focuses on measuring aspects of DQ that are concerned with "fact", rather than "opinion". For example, accuracy and currency of data are objective characteristics (a data item either is or is not 3 minutes out of date), while characteristics such as reputation and believability of data depend more on the eye of the beholder. One of the most common methods of objective DQ assessment is known as *database bashing* [Redman1996]. Essentially, this involves comparison of data in two or more databases that have some overlap in content. A typical application of the database bashing technique is to determine whether data quality levels are preserved through data feeds between systems. Database bashing can indicate where data fails to be consistent, accurate and complete, but the results must be interpreted with care, when for example inaccurate data is compared with inaccurate data or incomplete data with other incomplete data. It is also very expensive, and is too coarse-grained a method of measurement to really assist in the determination of root causes of data quality defects [Redman1996]. Database bashing is just one of a range of techniques often used in *data cleansing*; that is, the process of identifying and removing the defect from source data sets in order to prepare them for some new (unanticipated) use [Galhardas2001]. Other techniques used in data cleansing include the formulation of queries that search for hypothesised forms of defect, and manual inspection of samples of data or statistical summaries of data. While data cleansing can be very expensive in terms of staff time, it is often performed with only a minor impact on normal business processing activities, since it is common for data cleansing staff to extract relevant sets of data from the original source systems (e.g. overnight), which can then be examined and corrected entirely off-line [Hernandez1998].

If the objective of the assessment is to gain an overall picture of data quality levels relatively cheaply, then examining every piece of data in the system and every process in which that piece of data is used is overkill. Two approaches have been suggested for this purpose: *statistical sampling* [English1999] and *data tracking* [Redman1996].

The statistical sampling approach involves the extraction of a representative subset of the data from the system, which is then examined (manually or with the help of a querying tool) for defects. Once the quality levels of this much smaller subset have been identified, they can be used to estimate the quality levels present in the system as a whole. In order to ensure that the analysis of the sample accurately reflects the state of the total data population being assessed, it is necessary to select a sufficiently large proportion of the data for the sample, while minimising the cost of the assessment process. Furthermore, it is usually recommended that anyone following this approach should spend some time determining which parts of the system are likely to be the best candidate for DQ improvement before the sample is selected [English1999, Bowen1998].

Data tracking differs from this statistical approach, in that it attempts to directly assess the effects of processes on data defect levels, rather than concentrating on the data only. When data tracking is used, the modifications made to selected data items as they are affected by some business process are tracked, in as much detail as is practicable. Thus, not only can defect rates be estimated by measuring the defects introduced in the sample (tracked) population, but we also gain considerable insight into exactly which parts of the process are responsible for creating them. However, data tracking cannot be used to gain a quick measure of data quality levels, as it must be carried out alongside the normal business processes, and cannot be simply invoked on demand [Redman1996].

One common feature of many of the methods proposed for assessment of objective DQ is that queries are formulated which search for defects in the data sets to be assessed. This is similar in practice (though not in intent) to techniques for integrity checking, in which queries are used to search for violations of the constraints. However, the aim of data quality assessment is simply to record details of the defects present (albeit often with the intention that these defects will later be corrected), whereas integrity checking has a much stricter aim of refusing to allow any modifications to data which would result in a violation of the constraint. Traditional integrity checking is unsuitable as a means of assessing data quality levels for improvement programmes. Checking complex constraints is a very expensive process, which inflicts the heaviest cost when transaction processing rates are highest. Moreover, they require strict adherence to all the constraints that they enforce, which is very limiting in the context of most messy, real world systems, where many constraints will have rare but legitimate exceptions [Caine2001].

However, the techniques which underlie integrity checking can be useful for data quality assessment, if adapted and weakened to suit this new application. In the remainder of this paper, we describe how we have adapted techniques from distributed integrity checking and distributed query processing, in order to produce a system which is more suitable for continuous, non-intrusive assessment of organisational data quality levels.

3 Non-Intrusive Assessment of Organisational Data Quality

We have developed two special software components that can be added to an existing network of information systems, to provide the capability for non-intrusive assessment of data quality within that system. The assessment method is based on the repeated execution of queries that can detect the presence of defects, at times which have been specified in advance to correspond to "quieter" periods for the individual systems. Any results found by execution of these queries are timestamped and recorded in a dedicated database, for later analysis. The two component types are:

- The Local Quality Assessor component (LQA), which schedules the repeated execution of the query fragments that have been allocated to a specific site within the network, according to the workload patterns specified by the owner of that site. This component also handles transfer of intermediate and final result sets to other sites, as necessary.

- The Global Quality Assessor component (GQA), which allocates the work involved in the defect detection queries to the local sites, according to the information provided by the cost model. The GQA also manages the repository in which details of any defects found within the network of systems are recorded.

In general, there will be one GQA per network of information systems, and one LQA for each site within that network that is expected to be involved in data quality assessment activities. The resulting architecture is illustrated in Figure 1.[1]

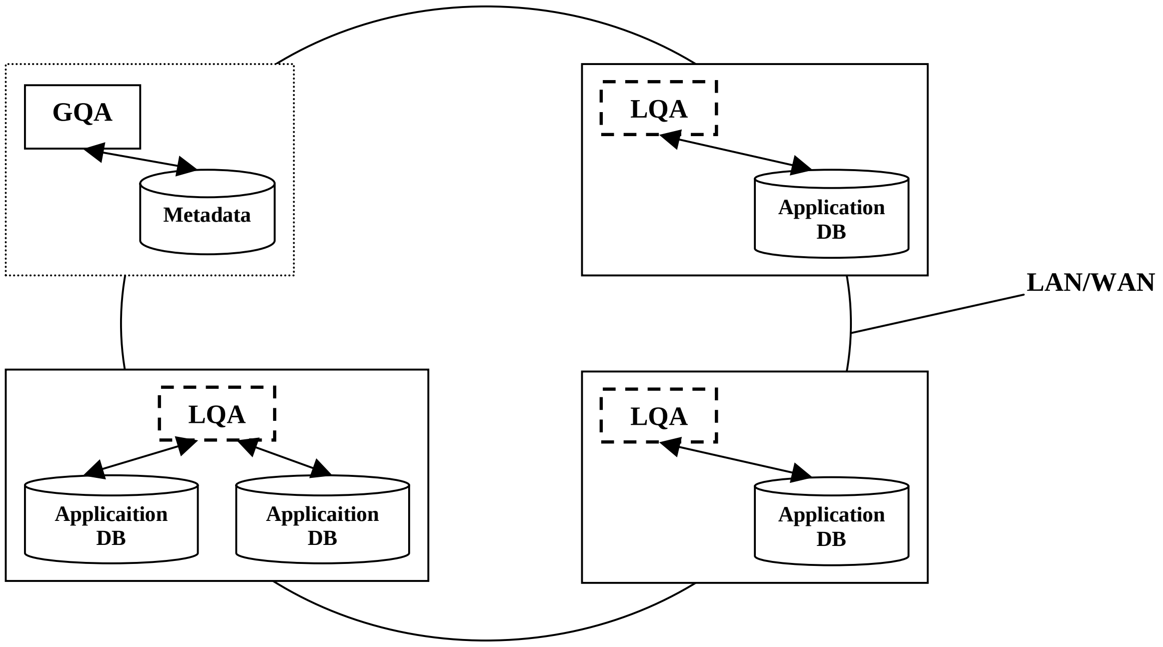


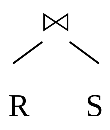
Figure 1. The System Architecture for Non-Intrusive DQA

[1] We also assume that all data sources within the network are accessible via the same SQL interface. Where individual sources do not provide this capability, it is assumed that appropriate wrappers will be provided, which will mimic a relational query interface for use by the LQAs.

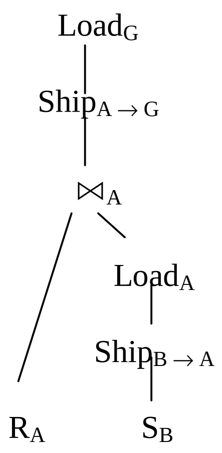
3.1 Allocation of Defect Detection Activity

Defect detection queries are specified as relational algebra expressions over a global schema that integrates the schemas of all the local systems. The relationship between the global schema and each local schema is defined by a series of mapping rules, and queries expressed against the global schema are transformed into equivalent queries over the local schema by "unfolding" the mapping rules within the query, as is common in distributed query processing [Ozsu1991]. This task is performed by the GQA, which also stores details of the data detection queries currently being monitored, the global and local schemas, the mapping rules and the data access characteristics of each local site. This latter set of information is used by the cost model to estimate how much time individual query fragments will take to execute at each local site, and includes such information as the block size of the local DBMS, relation sizes and selectivity statistics. Its use is discussed further in Section 4.

After transforming the query so that it references only local tables and attributes, the GQA fragments the query into a number of sub-queries, and allocates each sub-query to a particular local site. This produces a set of sub-queries which each access data that is local to one site or that has been shipped to that site from another. In general, there will be several possible allocations that can be made for any given query. For example, consider the following very simple query, which joins two tables *R* and *S*, where *R* is stored at site *A* and *S* is stored at site *B*:



There are three elements to this query (the two local relations and the joined result relation), and theoretically either site (*A* or *B*) could be given the task of executing the join operator, giving a total of two possible allocations. However, different allocations will result in different data transfer and load patterns. For example, if *R* is much larger than *S*, it will generally be more efficient to ship *S* to site *A* to compute the join, than to ship *R* to site *B*. The chosen allocation plan is indicated by inserting instructions to ship data, and to load it into local storage for further processing, into the original query expression. For example, we might produce the following allocation plan from the above query:



Notice that additional shipping and loading operations are added to the root of the allocation plan, in order to ensure that the final results of the query are shipped to the GQA, to be timestamped and logged in the database of detected defects. This occurs even if an empty result set is produced, as we need to record when defects have been corrected, as well as when they are introduced.

In traditional distributed query processing (and integrity checking) it is usual to try to choose an allocation that minimises the time spent transferring data between sites, as this is the major delaying factor in the evaluation. As we shall see in Section 4, the situation is not quite so simple for non-intrusive assessment of data quality, and a variety of other factors have to be taken into account. However, certain obviously stupid allocations can be ruled out of consideration straight away, and the GQA uses a couple of simple heuristics to avoid generation of such allocations:

- Always allocate production of a local relation to the site at which that local relation is stored.

- Always allocate a unary operator to the same site as its operand.

In addition to fragmenting each defect detection query, and allocating the sub-queries to the various LQA components for execution, the GQA also provides each LQA with some information about the order in which its query fragments must be executed. Largely, this order is based on the normal dependencies within the query, so that a sub-query which uses the result of some other sub-queries as its operand must be executed after that other sub-query. However, for complex queries, it is possible that two sub-queries may be independent of each other, but still be allocated to the same site. In these cases, the GQA tries to suggest the most efficient order for their execution, based on the cost model (to be described later).

3.2 Non-Intrusive Execution of Defect Detection Activity

Once a defect detection query has been allocated to the relevant local sites, the LQA components take over the task of repeatedly executing the sub-queries during times when this processing activity will not affect mission critical processing. The owner of each local information system specifies the expected workload pattern at that site in advance, in the form of a timetable showing predicted busy and idle periods. Figure 2 shows an example workload pattern. The times given in the workload are relative times, and the whole timetable describes a continuously repeating

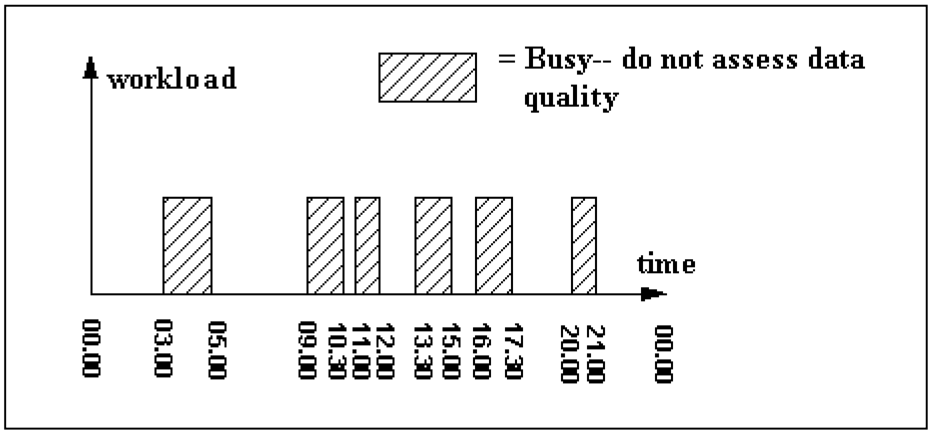


Figure 2. Example Timetable Showing Expected Local Workload Patterns

cycle. The workload pattern in this example would repeat every day, but timetables of arbitrary length may be specified (e.g. to capture the different workload patterns encountered in a typical week or month).

The internal structure of each LQA is shown in Figure 3. This architecture is similar to that used by the LOIS system [Caine2001], which checks integrity constraints in centralised systems in a non-intrusive manner. The LQA also contains a record of the sub-queries it is expected to evaluate (Operation Metadata), and any constraints on the order in which they should be evaluated, set by the GQA. Intermediate data sets, which have been shipped to this site from other LQAs, are also recorded, so that they can be loaded into the DB for further processing when required.

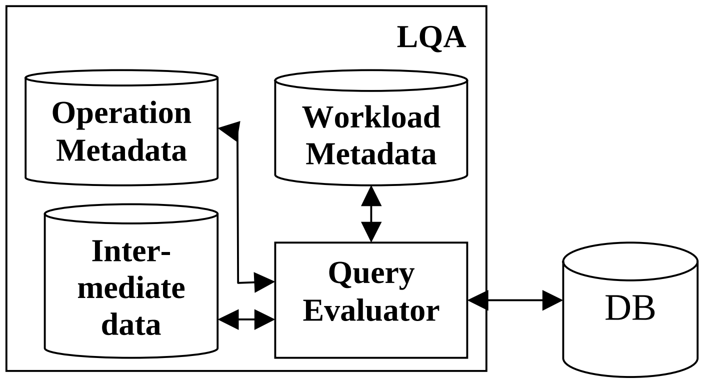


Figure 3. Local Quality Assessor Architecture

The query evaluator coordinates the execution of all the sub-queries that are the responsibility of each LQA. This component sleeps while the local information system is busy (as defined by the pre-specified workload) but wakes up when it enters a period of 'idle' time (or, alternatively, time when the system owner is prepared to devote some data processing resources to data quality assessment). It then repeatedly chooses a sub-query, data load operation or data ship operation from the Operation Metadata, and executes it, until the period of idle time ends. The exact criteria used by the LQA to select the next sub-query operation for execution is quite complex, and we can give only an overview of them here.

In designing the algorithm for selecting operations for execution, we were motivated by the following considerations:

- Since the time available for data quality assessment is necessarily limited, we wish to avoid wasting processing time when we know that we cannot produce any new information by the chosen action (e.g. by repeatedly executing a sub-query when its result cannot have changed).

- Most updates to data will occur during times when the system is "busy", but all query evaluation activity for data quality assessment must wait until the system becomes idle again. There will therefore necessarily be a delay between defects entering (or being removed from) the system and their detection by the data quality assessment system. We would like to minimise this delay as far as possible (in order to maximise the accuracy of the results).

In order to achieve the first of these aims, the LQA keeps track of when each sub-query result was last re-evaluated, when each local data set was last updated[2] and when each intermediate data set sent from another site is refreshed. This information allows us to determine the set of sub-queries that have not been evaluated since one or more of their operands was refreshed or updated. We say that these sub-queries are *eligible for execution*. All other sub-queries need not be considered, since no new information can be gained by evaluating them.

In order to achieve the second of our aims, we must find some way to choose one sub-query from those which are eligible for execution, so that the delay between changes to operands and evaluation of the sub-queries are minimised. We therefore choose to execute that sub-query which has the largest gap between refresh/update of one of its operands and the current time, subject to the ordering constraints imposed by the GQA.

By this means, the distributed defect detection query is executed (sometimes) concurrently by the LQAs, according to their local workload constraints. Each time execution of a query completes, the final result set (which may, of course, be empty) is sent to the GQA, where it is time-stamped and logged for future analysis. Thus, we can achieve "almost" continuous monitoring for defects, at a greatly reduced cost to the organisation than could be achieved with either full integrity checking or 'one-off' data cleaning attempts. In the next section, we will discuss whether the non-intrusive approach described here can provide information that is as useful as that provided by these other alternative approaches, and how its shortcomings can be addressed.

4 Addressing the Shortcomings of Non-Intrusive Data Quality Assessment

We have already discussed how in order to improve data quality within an organisation it is necessary to measure the initial quality levels, then to analyse the resulting measurements in order to decide what action to take to prevent more defective data from being introduced. However, if the data quality is not measured accurately enough, then the results may not reflect the real status of the organisation's data [English99]. Just as poor quality operational data can result in mistaken decisions being taken by managers, so inaccurate or incomplete measurements of data quality can mean that the actions instituted by data quality staff fail to achieve the desired improvements.

In order to achieve "perfect" knowledge of the data defects that are present in a system, and of when they entered and were removed from the system, it is necessary to monitor every single modification made to data and to compute its effect on the current set of data defects.[3] This is effectively what integrity checking does. Whenever an update occurs which may violate a constraint, a query is issued in order to determine whether that constraint is indeed violated [Embury1995]. We could adapt the same behaviour for data quality assessment, in order to

[2] We assume that the local information systems have a facility for informing the LQA when local data is updated. If the local system has a trigger mechanism, then this is easily achieved. If not, it may be possible to discover that data sets have been updated by examination of the transaction log or other metadata.

[3] Of course, our view of "perfection" here is relative to the set of defect detection queries we have formulated. If we formulate a query incorrectly, we will get inaccurate details of the defects that are present. Moreover if we fail to add a query for a particular type of defect that we are interested in, then we will obtain a highly incomplete picture of the prevalence of that kind of defect.

obtain fully accurate and complete knowledge of data defects. Unfortunately, however, the impact on normal business processing would be prohibitive. At the other end of the spectrum, we have one-off data cleansing type activities, in which the effect on normal business processing is extremely limited, but the information obtained is highly incomplete. For example, if data quality levels are only assessed every three months, then many defects may have entered and been removed from the system in the interval between assessments. These defects may be the cause of much lost revenue, but they are not considered by the improvement programme staff, as they are effectively invisible.

The non-intrusive method of defect detection described in the previous section represents a compromise between these two extremes. The cost in terms of data processing resources is much higher than for off-line data cleaning but is considerably less painful than the cost of integrity checking. In addition, because defect checking activities will be performed on (hopefully) a daily basis, the system is much more likely to detect the presence of short-lived but costly defects. The resulting knowledge of defect levels and types is therefore much more complete than under the data cleaning approach.

Unfortunately, we have introduced a new problem by delaying and distributing the defect detection queries: namely, inaccuracy of defect data. The problem is caused by the phenomenon of "phantom states" in distributed query processing [Grefen1997], in which defects may be incorrectly detected or ignored. These are caused by the fact that different parts of the data are checked for defects at different periods in time, and that intermediate results are cached throughout the system (for the purposes of comparison of data from different sites). The delays inherent in distributed query processing mean that these caches quickly become out of date.

For example, consider a defect detection query that finds all examples of "orders" where some product on the order is priced differently to the stated price in the product catalogue. The table *Orders(ProductID, ProductPrice)* is stored at site 1 and the catalogue table *Products(ProductID, ProductPrice)* is stored at site 2. The evaluation of this query can be expressed as three sub-queries:

- $P_{1:}$ ships the *Orders(ProductID, ProductPrice)* table from site1 from site2.
- P_2: executes the query:

  *select * from Orders, Products where (Orders.ProductID = Products.ProductID) AND (Orders.ProductPrice != Products.ProductPrice),*

  at site2.
- $P_{3:}$ ships the result of P_2 from site2 to the global site G.S.

The workloads at the two sites in the example system are shown in Figure 4. Assume that a new record is added to the *Orders* table at time T1. This record is in fact defective, but it is deleted from the system at time T3. However, because of the delays inherent in the non-intrusive execution of the query, the defect still appears to be present in the database at time T5. Moreover, we do not discover its disappearance until time T6.

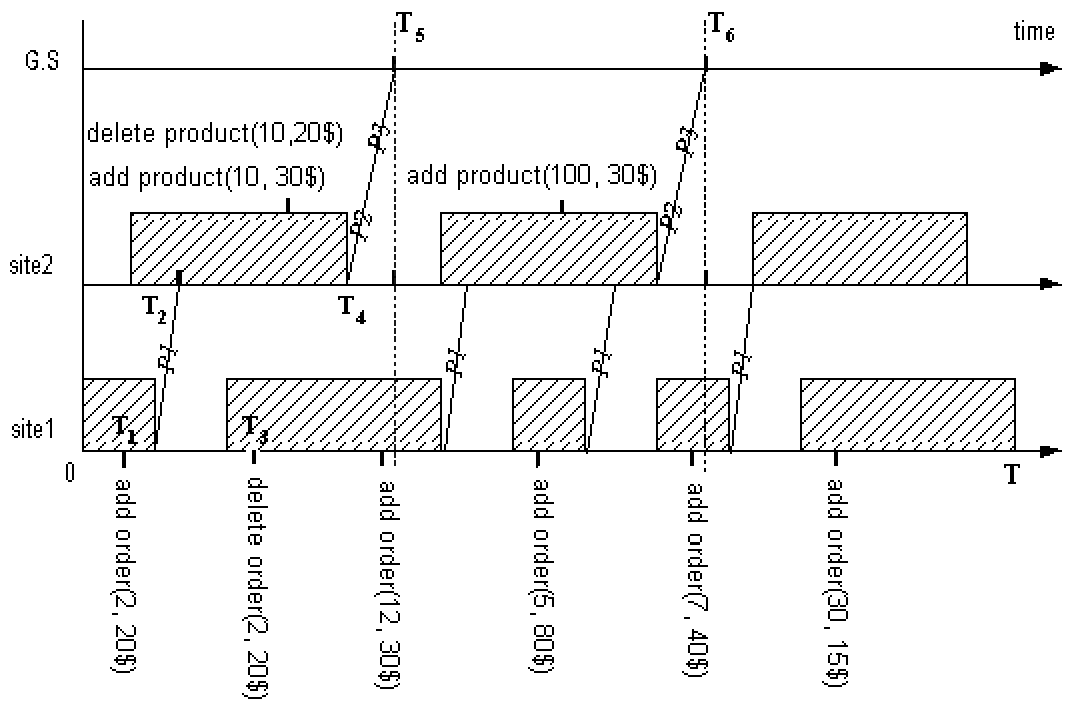


Figure 4. Example of Non-intrusive Quality Assessment Allocation 1

An alternative allocation of the work for this same defect detection query is given by the following three sub-queries:

- P_4: ships the *Products(ProductID, ProductPrice)* table from site2 to site1.
- P_5: executes the query:
  *select * from Orders, Products where (Orders.ProductID = Products.ProductID) AND (Orders.ProductPrice != Products.ProductPrice),* at site1.
- P_6: ships the result of P_5 from site1 to G.S.

The pattern of checking for this allocation is shown in Figure 5. A comparison of these two figures indicates the different patterns of defect detection that can be achieved simply by deriving a different allocation of work. For example, we can see that the defect enters the system at time t1, and is detected at time t2, thus suggesting that this allocation may result in more accurate results than the previous one. Moreover, we can see that this second allocation results in more frequent reports of defects than the first. We see that defects are reported on four occasions in Figure 5 (indicated by the dotted vertical line), but just two occasions in Figure 4. We could therefore prefer this second allocation to the first.

This second allocation may be more accurate than the first, but it does still have the potential for inaccuracy. Unfortunately, we cannot completely remove this inaccuracy without increasing the amount of data processing resources required for defect checking to unacceptable levels. However, we can attempt to make use of the resources available to us (as described by the local site workloads) in such a way that maximises the accuracy and completeness of the resulting data defect records. We have already seen how choice of allocation of sub-queries to the different sites in the system can affect the amount of time that must be spent transferring data from one site to another. Similarly, the fact that each site may be operating to a different workload pattern means that different allocations will result in more accurate or more complete defect detection patterns than others. We can therefore maximise the benefits to be gained from our defect detection activities, if we can find some means to choose the allocation that will result in the most accurate and complete defect reports possible.

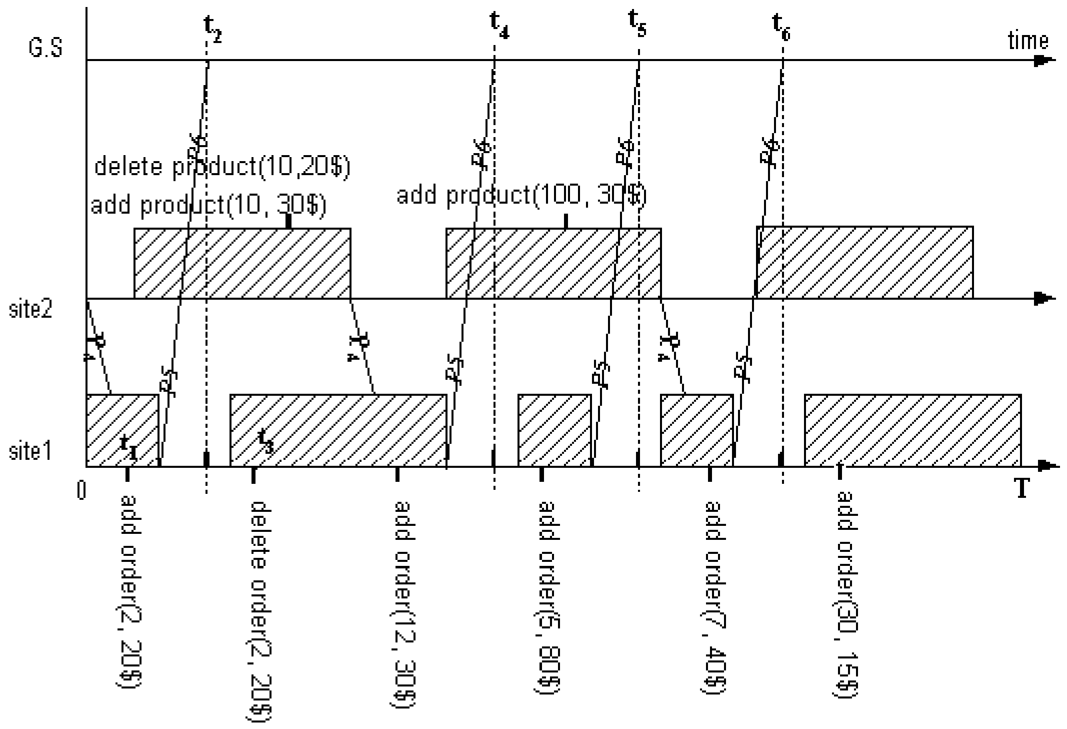


Figure 5. Example of the Non-intrusive Quality Assessment Allocation 2

In order to achieve this, we have developed a cost model which can be used to analyse the characteristics of different allocation plans, so that the most appropriate one can be selected. The cost model is based around a number of parameters. These will now be described before discussing how they can be used to determine the characteristics of a particular allocation.

The cost model simulates the behaviours of each individual allocation plan over a time period according to the workloads specified in advance. In order to reflect the workloads completely, the duration of this simulation period needs to be adequate. In a particular simulation period, if the defect checking query is executed more frequently, then the DQA is likely to be more accurate and to provide a more complete record of the defect data. We define the number of the times that the result of defect checking arrives at the global site during the simulation period as the 'Frequency' of DQA in this period. For example, in figure 5 mentioned previously, the 'Frequency' equals six during the period 0-T. In addition, for one allocation plan *AP1*, if the time interval between the evaluation of a sub-query and the corresponding update to one of its source relations is shorter than the time interval for another allocation plan *AP2*, then the assessment result of *AP1* is likely to be more accurate and complete than that of *AP2*. We call this characteristic the 'Delay'. To compute the delay inherent in the complete evaluation of some defect detection query, we sum the time intervals between the update to each of the source relations and the recording of the final query results at the global site. Clearly, due to the complex interactions of multiple workloads, some evaluations of the defect query will result in a larger 'delay' than others, even with the same allocation plan. In order to gain an impression of the overall level of delay caused by a given allocation plan, the average delay per complete defect check is computed for the entire simulation period. This is done using the following formula:

Inaccuracy = sum(Delay)/ Frequency

Thus, at compile time, the GQA uses the above cost model to select the most appropriate distribution of defect detection work, and hopefully to make sure that the defect detection queries assess DQ as accurately and as completely as possible given the available resources.

5 Conclusions

We have described an approach to the assessment of data quality across a network of linked information systems. This approach balances the value of the information obtained against the cost to the organisation of achieving this information bearing in mind limitation processing resources. In particular, we allow the owners of each individual system in the network to specify the expected workload on their system, and thus to define the times at which they are willing for data processing resources to be devoted to data quality assessment. Thus, we can achieve regular checking for defects with minimal disruption to crucial revenue-generating data processing activities. While this approach unavoidably introduces elements of inaccuracy and incompleteness into our resulting knowledge of defect levels, we have shown how a more sophisticated cost model can help to minimise these elements, so that maximum benefit can be gained from the small amount of data processing resource given over to DQ assessment.

The next step in our work is to attempt to validate and refine our cost model, through experimentation. We are also working on the development of a range of visualisation tools which can be used to analyse the detected defect data, in order to assist in the discovery of the root causes of defect classes [Baillot2001]. The design of these tools is made non-trivial by the need to cope with the inherent inaccuracies within the defect detection process, and we plan to make use of techniques from temporal logic to extract weaker, but more reliable knowledge from our defect logs. We hope that these tools will be of particular value in allowing defect causal analysis techniques [Card1993] to be adapted for use with data defects, as well as in underpinning the design and monitoring of data quality improvement programmes.

References

[Baillot2001] A.Baillot. Visualising Data Quality Problems. *MSc thesis* (in preparation), the Department of Computer Science, the University of Manchester, UK, 2001.

[Bowen1998] P.L. Bowen. *Continuously Improving Data Quality in Persistent Databases*. http://www.dataquality.com/998bowen.htm.

[Caine2001] N.J. Caine and S.M. Embury. *LOIS: the "Lights Out" Integrity Subsystem*, in Proceedings of 18th British National Conference on Databases (BNCOD'01), B.J. Read (ed.), Chilton, UK, July, Springer LNCS V. 2097, pp. 57-74, 2001.

[Card1993] D.N. Card. Defect-Casual Analysis Drives Down Error Rates. *IEEE Software*, 10(4), pp. 89-100, July 1993.

[Chawathe1996] S.S. Chawathe, H. Garcia-Molina, and J.Widom. *A Toolkit for Constraint Management in Heterogeneous Information Systems*, in Proceedings of the 12th International Conference on Data Engineering, S.Y.W. Su (ed.), New Orleans, Louisiana, IEEE Computer Society, pp. 56-65, 1996.

[English1999] L.P. English. *Improving Data Warehouse and Business Information Quality: Methods for Reducing Costs and Increasing Profits.* Wiley Computer Publishing, New York, USA, 1999.

[Embury1995] S.M. Embury and P.M.D. Gray. *Compiling a Declarative High-Level Language for Semantic Integrity Constraints*, in Proceedings of the 6th IFIP TC-2 Working Conference on Data Semantics (DS-6), R. Meersman and L. Mark (eds.), Stone Mountain, Georgia, USA, Chapman and Hall, pp. 188-226, 1995.

[Galhardas2001] H. Galhardas, D. Florescu, D. Shasha, E. Simon and C. Saita. *Declarative Data Cleaning: Language, Model and Algorithms*, to appear in Procedings of the 27th International Conference on Very Large Data Bases, Roma, Italy, Morgan Kaufmann, 2001.

[Grefen1997] P.W.J. Grefen and J. Widom. *Protocols for Integrity Constraint Checking in Federated Databases*, Distributed and Parallel Databases 5(4), pp. 327-355, January1997.

[Haas1997] L.M. Haas, D. Kossmann, E.L. Wimmers, and J. Yang. *Optimizing Queries across Diverse Data Sources*, in Proceeding of the 23rd VLDB Conference, M. Jarke, M.J. Carey, K.R. Dittrich, F.H. Lochoysky, P. Loucopoulos, M.A. Jeusfeld (eds.), Athens, Greece, Morgan Kaufmann, pp. 276-285, 1997.

[Hernandez1998] M. Hernandez and S. Stolfo. *Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem*, Data Mining and Knowledge Discovery 2(1), pp. 9-37, January 1998.

[Huang1999] K.T. Huang, W.L. Yang, and R.Y. Wang. *Quality information and knowledge*. Prentice Hall PTR, New Jersey, USA, 1999.

[Nicolas1982] Jean Marie Nicolas. *Logic for improving integrity checking in relational databases*, Acta Informatica, Volume18, pp. 227-253, 1982.

[Oszu1991] M.T. Ozsu and P. Valduriez. *Principles of Distributed Database Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1991.

[Redman1996] T.C. Redman. *Data Quality for the Information Age.* Artech House, Boston, USA, 1996.