Checkpoint-based resumption in data warehouses

Marcin Gorawski and Pawel Marks

Silesian University of Technology,
Institute of Computer Science,
Akademicka 16,
44-100 Gliwice, Poland
{Marcin.Gorawski, Pawel.Marks}@polsl.pl

Abstract. In the paper we focused on the problem of efficient handling of ETL processes failures. During such a process, a data warehouse is filled with data. Because large amounts of data need to be processed, the whole process takes a lot of time. After a failure there may be no time to restart the process. In such a situation a resumption algorithm should be applied. In the paper we present a new approach to the checkpoint-based resumption method. We combine checkpointing with the Design-Resume algorithm. Such a combination is supposed to work more efficiently than the pure checkpointing. Moreover, not all the ETL application modules must implement the checkpointing. We present a basic idea of the algorithm, its requirements and necessary definitions. The proposed solution is then compared to other resumption methods and obtained results are discussed.

1 Introduction

Data warehouses collect large quantities of data. Their task is to provide the decision support applications used by managers and directors with the necessary data. The more up-to-date the warehouse is, the closer to the reality are the results of analysis performed by DSS applications, and the better decisions can be made. The data set stored in a data warehouse (DW) is usually taken from transactional systems. Not all the data are required, in business applications it is usually approximately 20% of the transactional data set. Moreover, records are usually processed before they are loaded into a destination database. A whole process of extracting and transforming the data and loading them into a destination is called ETL that is an abbreviation for *Extraction, Transformation and Loading.*

Nowadays data warehouses collect giga- or even terabytes of data. It is not a surprise that in the case of so huge data sets, an ETL process (further called simply *extraction process*) takes long hours or even days to perform a full load. Depending on a data warehouse system two kinds of DW loads may be met: full and incremental. During a full load all the data already stored in a DW are deleted, and when the warehouse is empty then a loading starts. During incremental load only the data that changed since the last load are processed. It makes the incremental load much shorter than the full load. However, it is not always possible to run the incremental load. If the

data changes are too complex, or aggregates computation is made in a way they cannot be easily updated, we must run a full load.

During a load process, a data warehouse is not available. That is why an ETL process is usually run in a time window when the system is idle (for example in the night or during a weekend). Here a problem may appear. In order not to disrupt the managers' work, an ETL process must not exceed the fixed time window. It fits the window if the processing goes without any unpredicted events. Unfortunately such an ideal situation sometimes does not take place. Statistically every thirtieth ETL process fails due to a system or hardware failure [10].

Occurrence of a failure interrupts an ETL process. The warehouse contains partially loaded data set, which in most cases is inconsistent. Such a dataset is unusable. In such a case we have three choices:

- restart the ETL process,
- restore the warehouse from a backup copy created before starting the ETL process,
- run the resumption procedure and continue the interrupted process.

Restarting the extraction from the beginning is the easiest option but it is also the most inefficient. The second option is to use a backup copy of the warehouse content. It is better to have the old data than to have no valid data at all. The best choice is to load only the missing part of the data set. The data set will be then consistent, and the time of producing the missing tuples should be short enough to fit in the remaining time. Resuming the interrupted ETL process is called resumption process.

In this paper we focus on the algorithms for resumption of the interrupted extraction process. In section 2 we present the current state of the art, describe the most common resumption techniques, present their advantages and disadvantages. In section 3 we present our approach to the checkpoint-based resumption algorithm. The results of the performed tests are included in section 4.

2 Previous works

Most commercial tools or tools such as Ajax [3] do not consider the internal structure of transformations and the graph architecture of ETL processes. Exceptions are researches [11, 12], where the authors describe the ETL ARKTOS (ARKTOS II) tool. To optimize ETL process, there is often designed a dedicated extraction application adjusted to requirements of a particular data warehouse system. Our experience prompted the decision to build a developmental ETL environments using JavaBeans components (ETL/JB and DR/JB). In the meantime, a similar approach was proposed in paper [1].

Further speeding up of the ETL process encouraged us to abandon JavaBeans platform. Our new ETL-DR environment succeeds the previous ETL/JB (JavaBeans ETL environment) and DR/JB (ETL environment with a Design-Resume algorithm [7]). The new ETL-DR environment is a set of Java object classes, used to build extraction and resumption applications. This is analogous to JavaBeans components in the DR/JB environment. In the DR/JB we implemented a dynamic estimation mechanism detecting cases when the use of DR resumption is inefficient. Another direction of our research is combining the DR resumption with techniques such as staging and checkpointing.

Similar research was presented in [7] where the authors compared the DR algorithm to its combination with savepoints. They proved that the DR-savepoint combination performs a little better than the pure DR. Unfortunately these experiments were performed on very small TCP-D data sets. In our opinion it gives non-representative results, because in data warehousing we have to deal with much larger datasets. We tried to combine the DR algorithm with the staging. After a failure part of the data set can be restored from the disk and there is no need to process it again. We gave the name "hybrid resumption" to the obtained algorithms combination. In [5] we showed that the proper use of the staging can be a quite efficient solution. The proper selection of the nodes writing stage data to disk files is crucial to reduction of the overhead imposed on the uninterrupted extraction process. In our experiments we managed to increase the resumption efficiency whereas the normal extraction time remained almost unchanged.

3 Checkpointing

A concept of *checkpoints* or *snapshots* is in general very simple. Assuming that there is a process running for a long period of time and it is failure prone (mostly hardware failures) we can create so called checkpoints. What a checkpoint is? It is nothing more than a copy of the process state. It is saved in a way that makes it possible to revert the process to the saved state and continue the processing whenever there is such a need. There are many applications of this method: fault tolerance, process migration, job swapping [9], virtual time [2].

We decided to apply the checkpointing to increase the resistance of the data extraction process against system failures. The extraction process usually takes a lot of time and cannot be interrupted. Accidental hardware failure or blackout may lead to loss of the results of many hours of work, even if it is was very close to the end of the process.

Our previous experience with combining the Design-Resume algorithm (DR) with the staging technique prompted us to combine the DR algorithm with the checkpointing. Both the DR and checkpoints are very efficient methods. The difference between them is the overhead imposed on the normal extraction process. The DR takes no additional actions during the extraction, so it has no influence on the process duration. The checkpointing is completely different. It may lengthen the processing even a few times depending on the frequency of checkpoints creation and amount of the data stored during each save.

Increasing the frequency of checkpoints creation leads to significant drop of the processing efficiency. In our research we focus on creating checkpoints in the most efficient possible way. We want to combine the checkpoints with the DR algorithm which uses the graph-based ETL process description, so in our research we will use the graph description also.

3.1 Graph-based ETL process description

In graph representation of the ETL process, graph nodes are responsible for tuples processing and graph edges define tuple flow directions. Each node belongs to one of the three categories:

- extractors reading data from sources

Fig. 1. Example of a simple extraction graph. E is an extractor node reading data from a source. T stands for a transformation node, such as filtration, grouping etc. I is an inserter loading data to a destination (e.g. a database table)

- transformations performing operations such as selection, projection, aggregation, etc.
- inserters loading tuples to destination places

An example graph is presented in Fig. 1. It consists of one extractor, one inserter and a few transformations. It is an example of the linear processing, it means that each graph node has at most one source node and one target node. Of course, the graph can be much more complicated: transformations can receive data from many source-nodes (joins, for example) and send results to many target-nodes.

The DR algorithm requires that the extraction graph is acyclic: during traversing the graph, there is no possibility to visit the same node twice. In our research this limitation is not a problem, it even simplifies graph analysis.

Each graph node and each node input is described by a set of boolean properties and key attributes. These properties are used by the DR algorithm to compute the place and type of additional filters used during resumption. Thanks to these properties, the algorithm can treat the nodes as black boxes and does not have to know anything more about the processing perform by them. The great advantage of the DR algorithm is no need of modifications of the existing nodes. During resumption they remain unchanged, only additional filter nodes are inserted into the graph to ensure that only the missing part of the data set will be produced.

3.2 Details of the ETL process implementation

To talk about the optimization of the checkpoints creation we have to provide some details of the ETL application implementation. The graph-based ETL process is implemented as a multi-threaded Java application. Each node works as a separate thread communicating with other nodes via shared memory and message passing.

Fig. 2 shows an example of connection between two nodes. Node 123 produces tuples and stores them in the output buffer. The buffer is a multichannel structure; it can transmit data to multiple receiver-nodes. Such a receiver in this case is node 124. It has an input parameter defined as a source node ID = 123 and output channel number = 1. Output buffer contains a packet queue in each logical channel, and each packet contains a limited number of tuples. Grouping tuples into packets increases the efficiency of communication between nodes by reducing the number of required thread synchronizations.

As can be noticed, one of the data sets that each node owns, is the output buffer containing output tuples. Moreover, there are nodes such as grouping ones, which can store in memory their temporary data structures. Depending on the kind of the processing it can be a small or quite large set of data. No matter when the checkpoint is created, all the data of each graph node must be saved. Besides the data sets, thread state has to be saved also, to enable restarting the processing from the point saved in the checkpoint.

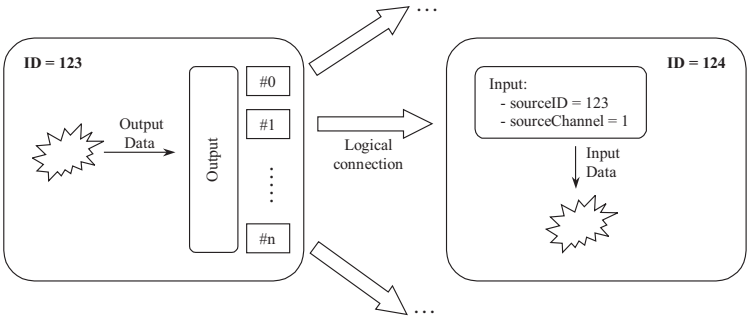

Fig. 2. Nodes interconnection on the implementation level. Data produced by node 123 are stored in a multi-channel output buffer. Source of the node 124 is defined as a node with ID = 123 and logical output channel number = 1

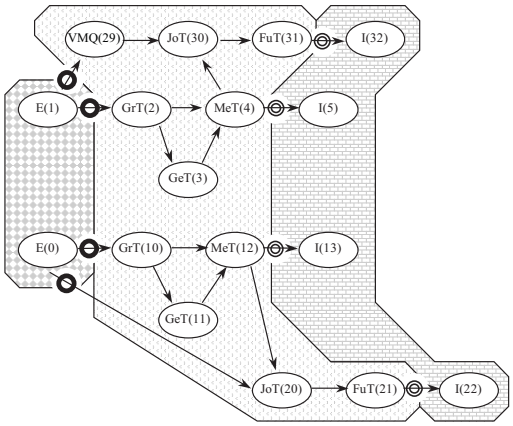

Fig. 3. Extraction graph divided into three functional blocks which are: extractors (the left most), transformations (in the middle) and inserters (the right most)

3.3 Graph analysis

We propose dual approach to the graph analysis for checkpoints. First, the graph is analysed as in the DR algorithm [7]: the nodes properties are processed. In the next step, the graph is seen as three functional blocks: extractors, transformations and inserters. Between these blocks the connections exist: many connections from extractors block to transformations block, and as many connections from transformations to inserters as many inserters there are in the graph.

Fig. 3 presents a graph split into functional blocks. Connections between the blocks are marked with circles. In the given example it would be the best to be able to save the state of all the nodes belonging to the three functional blocks. In practice it is usually impossible or it costs too much time:

– each inserter would have to make a copy of a complete database table. In data warehousing it means transferring even gigabytes of data which takes a very long time

- period of time between checkpoints could be treated as a transaction but even assuming that the database could handle such a case, a synchronization of such a transaction in distributed data warehouse would be too complicated (if at all possible)
- extractors would have to be able to return to the place in the data stream where they were when the checkpoint was created. It is possible to do, but requires additional implementation-level modifications

However, there are no significant difficulties to save the state of transformation nodes. This is the main assumption, the algorithm discussed below bases on.

3.4 Algorithm details

The goal of the presented algorithm is periodical saving of states of the transformation nodes and optionally extractors or inserters. The saved states should enable resumption of the interrupted ETL process. General steps of the algorithm are as follows:

1. Analysis of the graph properties to check if the algorithm can be applied
2. Periodical creation of the checkpoints
3. Assignation of filters for resumption phase
4. Insertion and initialization of resumption filters, optional switching inserters/extractors into resumption mode

Graph traversing and analysing The graph is analysed as in the DR algorithm in both topological and reversed direction. In the first step graph nodes are checked if they support checkpointing:

- a transformation should have the possibility to save and restore its state in any moment of the processing,
- an inserter should have the possibility to save and restore its state or possibility of identification of the last loaded tuple,
- an extractor should have the possibility to save and restore its state or possibility to get the part of the stream read before a failure once again,

Further in the paper we will say that if a node can save and restore its state, it holds the $checkpointable$ property. Now for each graph node X, a transitive property *checkpointFeasible(X)* is computed. It is defined as follows:

Definition 1. *checkpointFeasible(X) = true, if:*

- *X is an inserter and holds checkpointable property or (it holds suffixSafe*[1] *and mapToOne*[2] *properties and can remember the last loaded tuple)*
- *X is a transformation and holds checkpointable property and all its direct target nodes holds checkpointFeasible property*
- *X is an extractor and all its direct target nodes holds checkpointFeasible property*

[1] $suffixSafe$[7] property is described in the definition 4

[2] $mapToOne$[7] property is held if each input tuple contributes to no more than one output tuple. All inserters hold this property

If any of the extractors does not hold the *checkpointFeasible* property, it means that the checkpointing cannot be applied to this graph, because one or more nodes do not support the method.

In the second step the graph is analysed to check the possibility of additional filters insertion. The task of an additional filter is removing from a tuple stream these tuples which were processed before creating the checkpoint (extractor filters) or tuples loaded after checkpoint creation and prior to a failure (inserter filters). Such filters can (but do not have to) be placed on connections between the three functional blocks. Here we can distinguish two cases:

- extractor does not hold *checkpointable*. The filter is inserted just behind the extractor
- inserter does not hold *checkpointable*. The filter is inserted just in front of the inserter

In the Design-Resume algorithm[7] four types of filters are used: CleanPrefix, CleanSubset, DirtyPrefix, DirtySubset. We focus only on the two first filters, which were described in details in [7].

Filters preceding inserters are inserted according to the rules known from the DR algorithm. Both CleanPrefix and CleanSubset filtration is possible. CleanPrefix filter requires the input of the inserter Y connected to the node X to hold the transitive property *sameSuffix(Y_X)*. This property denotes that on the input Y_X the suffix of the data stream will be provided in the same form as it would be provided if the processing would not have been interrupted. If this property is not held, only a CleanSubset filter can be used. The *sameSuffix* property bases on the following properties:

Definition 2. *sameSet(X) = true, if:*

- *X is an extractor and during the resumption it generates the same set of tuples as prior to a failure*
- *X is a transformation that for the same input sets always generates the same output set*

Definition 3. *setToSeq(Y_X) = true, if for any permutation of the input set received from the node X, the node Y always generates the same output sequence. It is true for sorting transformations.*

Now a *sameSuffix* property can be defined:

Definition 4. *sameSuffix(Y_X) = true, if:*

- *X is an extractor and during the resumption it generates the sequence of tuples as prior to a failure. Optionally a prefix of the sequence excluding the last prefix tuple can be removed*
- *X is a transformation that holds inDetOut*[3] *property and whose each input node V holds sameSuffix for each input or (it holds sameSet(V) and the X's input holds setToSeq(X_V))*

[3] $inDetOut$[7] property is held if for the same input sequences the node generates the same output sequence

Saving application state When a checkpoint is created, state of all nodes holding *checkpointable* is saved. These are for sure all the transformations and optionally extractors or inserters. Efficient creation of checkpoints requires a special creation procedure. Out of the discussion is the necessity of putting the nodes into the state to which they can return after restoring their states. We talk about both the temporary data and buffers and also current position in the running code. Creation of a checkpoint has been divided into three phases: stopping the nodes threads in conjunction with output queues emptying, saving nodes state, continuation of the processing.

Filter assignation Behind an extractor only a CleanPrefix filter can appear. This guarantees that only the required data stream suffix will be provided to the transformation input. Of course the input must have key attributes set, otherwise filtration will not be possible. If an extractor supports such a possibility, the built-in reextraction procedures can be used instead of filters. Known from the DR algorithm *GetSuffix* procedure replaces the CleanPrefix filter, and *GetDirtySuffix* procedure can be used instead of the DirtyPrefix filter [7]. In this case additional CleanPrefix filter is still necessary to get the same result as by use of a GetSuffix procedure. Filters placed in front of inserters are being assigned basing on the DR algorithm rules.

The filters are required only when a particular extractor or inserter does not hold a *checkpointable* property. If the extractor or inserter state can be restored from a previously created checkpoint, no filters are needed.

Resumption initialization Resumption initialization procedure is simple. After an interruption of the ETL process, the latest checkpoint must be found. Then it is loaded which means that the state of all graph nodes holding *checkpointable* property is restored. Now insertion of additional filters begins and the filters are initialized. Filters inserted behind extractors are informed what the last tuple received by the subsequent transformation is. Filters preceding the inserters are initialized with data taken from the inserters. Next inserters are switched to resumption mode. It causes that already loaded data set is not erased and new tuples are appended to the existing set.

4 Efficiency tests

4.1 Test Conditions

The base for our tests is an extraction graph containing 4 extractors reading tuples from 4 source files and 15 inserters (loading nodes) loading tuples into 15 database tables. The graph consists of three independent parts, but it is seen by the extraction application as a single ETL process.

The ETL process generates a complete data warehouse structure. It is a distributed spatial data warehouse system designed for storing and analyzing a wide range of spatial data [4]. The data is generated by media meters working in a radio-based measurement system. All the data is gathered in a telemetric server, from which it can be fetched to fill the data warehouse. The distributed system is based on a data model called the cascaded star model. The test input data set size is 500MB.

Method	Extraction time [s]	% change to the fastest method
Hybrid	1475	+8%
DR stream	1366	0%
Checkpoint	1496	+9%

Tab. 1. Measured extraction time for failureless cases

The tests were divided into three parts. In the first part we examined the resumption efficiency of the hybrid resumption algorithm (DR + staging) [5]. During this test all the join transformations worked in a buffering mode. It means that they collect all the tuples from the slave input first, then they start on-line processing of the master input. In such a mode VMQ[4] nodes are required to avoid data flow deadlocks, but on the other hand we can make use of the staging technique used by the hybrid algorithm.

In the second step we analysed the efficiency of the pure DR algorithm. This time all the join transformations worked in stream mode. In the stream mode we cannot distinguish prebuffering and on-line processing phases. Tuples from both inputs are processed simultaneously and only a small set of tuples may be buffered. In this mode the VMQ nodes are disabled.

In the third part we used the same extraction graph as in the second part, but this time we examined the efficiency of checkpointing. We focused on both: increase of the normal processing time caused by checkpoints creation and the resumption efficiency. In this test the extractors and inserters were unable to remember their states in created checkpoints, so additional filters had to be inserted into the graph.

The tests were ran on two PC machines with Pentium IV processors and 512MB of RAM. On one of them Oracle10g database was running, and on the other one the ETL application was started. Communication with the database was implemented using Oracle OCI drivers and SQL*Loader. A single uninterrupted extraction process time varied from 22 to 25 minutes.

During each loading test the extraction process was interrupted in order to simulate a failure. The resumption process was then run and the time was measured. Using collected results we prepared resumption charts showing the resumption efficiency depending on the time of a failure.

4.2 Extraction and Resumption Tests

The goal of the tests is to compare the efficiency of various extraction and resumption methods for the same extraction graph. Three aspects were analysed: influence of the chosen method on the time of the normal (uninterrupted) ETL process, resumption time and overall processing time[5].

[4] *VMQ* (Virtual Memory Queue) is a special node storing large amounts of data on external storage to avoid running out of memory. It is desirable not to use it because accessing external storage lowers the efficiency

[5] Overall processing time is the sum of the resumption time and the time of failure. It expresses the amount of time required to finish the ETL process in case of failures

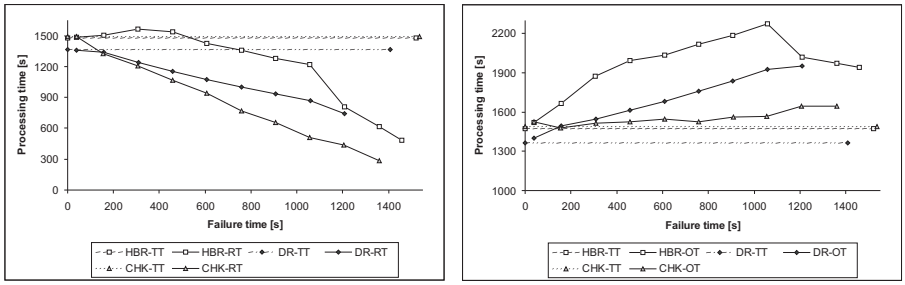

Fig. 4. Resumption (left) and overall (right) time plot. TT denotes *Total Time* of the normal processing, RT is the *Resumption time*, OT is the *Overall Time*

Table 1 compares extraction times for the three presented methods. The pure DR algorithm using stream-like joins is the fastest one. The reason why it is faster is in our opinion no VMQ nodes buffering the data. Data streams are processed on-the-fly, all the graph nodes are working all the time, none of them is idle. Unfortunately in such a case the use of staging technique makes no sense[5]. When join transformations work in a buffering mode, the data provided to the master inputs must be buffered until all the slave input tuple are read. The buffering lengthens the processing due to additional disk accesses. The third method uses the extraction graph used in the first test case, but it creates a checkpoint every 60 seconds. Creation of checkpoints lengthens the processing time also, but comparing to the hybrid method the overhead is relatively small.

Figure 4 shows the resumption times of the examined methods compared to the time of uninterrupted extraction. As we can see DR and checkpoints resumption efficiency is initially similar, but the later a failure occurs, the more efficient the checkpoint method is. For failures occurring at the end of the ETL process, the efficiency of the DR closes to the efficiency of the hybrid algorithm. The most important here is to have the resumption curve below the line denoting normal extraction time (TT). Otherwise it means that the resumption last longer than simple restarting the whole process from the beginning.

In fig. 4 overall processing times are compared also. Overall processing time is the sum of the processing time prior to a failure and the sum of the resumption time. It simply can be explained as the time between starting and finishing the ETL process assuming that after a failure the resumption process runs without any delay. Here again we can see that checkpointing is the best solution. The closer to the TT line is the resumption curve, the better. One should notice that it is impossible to have resumption curve below the TT line. If it was, it would mean that it is better to interrupt the extraction and then run the resumption.

5 Conclusions

In the paper we presented a new approach to the problem of resumption of the interrupted extraction process using checkpoints. The approach mixes two mechanisms: ETL application state saving and restoring which is typical for checkpointing, and usage of additional filters which is used in the Design-Resume[7] algorithm. We assumed that nodes such as extractors or inserters can work in a way making impossible to save and

restore their states. Even if it is possible, its overhead may be too big and may lower the efficiency of the running process significantly. The proposed algorithm can work without storing the states of extractors and inserters. To make the application consistent during the resumption additional DR-like filters are inserted into the graph. The task of the filters is to remove from a tuple stream these tuples which were processed before creation of the checkpoint or were loaded by inserters. In this case data loaded after creation of the checkpoint are not lost, and additional filter ensures that they are not loaded to the warehouse again.

The proposed solution was tested in the ETL-RT extraction environment implemented in Java for research requirements. The environment supports various resumption algorithms: the Design-Resume algorithm, staging technique, hybrid algorithm (DR + staging) and the presented checkpoint-based resumption. Because all these algorithms are implemented in the same environment, the results of the tests we obtained are reliable and valuable.

The results are very encouraging and promising. The time of the normal extraction process was increased by less than 10% in comparison to the fastest tested method. In exchange for this we obtained a significant resumption efficiency growth that was presented in figure 4.

References

1. Bruckner R., List B., Schiefer J.: Striving Towards Near Real-Time Data Integration for Data Warehouses. DaWaK 2002.
2. FujiMoto R.: Parallel discrete event simulation, Communications of the ACM, 33(10), 1990
3. Galhardas H., Florescu D., Shasha D., Simon E.: Ajax: An Extensible Data Cleaning-Tool. In Proc. ACM SIGMOD Intl. Conf. On the Management of Data, Texas (2000).
4. Gorawski M., Malczok R.: Distributed Spatial Data Warehouse Indexed with Virtual Memory Aggregation Tree. 5th Workshop on Spatial-Temporal DataBase Management (STDBM_VLDB'04), Toronto, Canada 2004.
5. Gorawski M., Marks P.: High Efficiency of Hybrid Resumption in Distributed Data Warehouses. 1st Intl. Workshop on High Availability in Distributed Systems (HADIS 2005), Copenhagen, Denmark 2005.
6. Gorawski M., Chechelski R.: Spatial Telemetric Data Warehouse Balancing Algorithm in Oracle9i/Java Environment, Intelligent Information Systems, Gdansk, Poland, 2005.
7. Labio W., Wiener J., Garcia-Molina H., Gorelik V.: Efficient resumption of interrupted warehouse loads. SIGMOD Conference, 2000.
8. Labio W., Wiener J., Garcia-Molina H., Gorelik V.: Resumption algorithms. Technical report, Stanford University, 1998.
9. Plank J. S., An Overview of Checkpointing in Uniprocessor and Distributed Systems, Focusing on Implementation and Performance. Technical report, University of Tennessee. 1997
10. Sagent Technologies Inc.: Personal correspondence with customers.
11. Vassiliadis P., Simitsis A., Skiadopoulos S.: Modeling ETL Activities asGraphs. InProc. 4th Intl. Workshop on Design and Management of Data Warehouses, Canada, (2002).
12. Vassiliadis P., Simitsis A., Georgantas P., Terrovitis M.: A Framework for the Design of ETL Scenarios. CAiSE 2003.