A MAPREDUCE FRAMEWORK FOR CHANGE PROPAGATION IN GEOGRAPHIC DATABASES

Ferdinando Di Martino, Salvatore Sessa
Dipartimento di Costruzioni e Metodi Matematici in Architettura, Federico II University
Via Monteoliveto 3, 80134 Napoli, Italy

Giuseppe Polese and Mario Vacca
Dipartimento di Matematica e Informatica, Salerno University, Via Ponte don Melillo, 84084 Fisciano (SA), Italy

Keywords: Change propagation, Geographic databases, Instance update language, Mapreduce, Schema evolution, Spatial datasets.

Abstract: Updating a schema is a very important activity which occurs naturally during the life cycle of database systems, due to different causes. A challenging problem arising when a schema evolves is the change propagation problem, i.e. the updating of the database ground instances to make them consistent with the evolved schema. Spatial datasets, a stored representation of geographical areas, are VLDBs and so the change propagation process, involving an enormous mass of data among geographical distributed nodes, is very expensive and call for efficient processing. Moreover, the problem of designing languages and tools for spatial data sets change propagation is relevant, for the shortage of tools for schema evolution, and, in particular, for the limitations of those for spatial data sets. In this paper, we take in account both efficiency and limitations and we propose an instance update language, based on the efficient and popular Map-Reduce Google programming paradigm, which allows to perform in a parallel way a wide category of schema changes. A system embodying the language has been implementing.

1 INTRODUCTION

Updating a schema is a very important activity which occurs naturally during the life cycle of database systems, due to different causes, like, for example, the evolution of the external world, the change of user requirements, the presence of errors in the system. When a schema evolves, there are many problems to cope with, like the change propagation problem (the problem of updating the ground instances of the database after a schema change occurs) (Ram and Shankaranarayanan, 2003). In the last years, the amount of spatial data has increased, thanks to technologies to capture spatial data (GPS or the satellites), so that systems using spatial data, like Geographic Information System, widening their application areas, has risen the increasing interest of institutions. However, once a spatial dataset has been designed and implemented, similarly to almost all databases, it has to be maintained and this entails the need to cope with schema changes. Therefore, spatial datasets require tools of schema evolution like all the other databases: in fact, even if some GIS tools (like, for instance, ESRI utilities (Arctur and Zeiler, 2004; Zeiler, 1999; Twumasi, 2002)) manage some schema changes, database administrator are often compelled to work without using any tool. This is due to the fact that existing tools manage only the so called simple schema changes (Banerjee et al., 1987) which often result insufficient to operate in real situations, as many researchers of schema evolution stressed (Lerner, 2000; Brèche, 1996). The lack of complete tools can make db administrators working directly on the geodatabase, making the changes directly on the physical database with a significant increase of the risk of creating a displacement between the conceptual model and the physical data. Designing and building a tool for schema evolution in spatial datasets is an important and challenging task, because changes in geographic databases can provoke significant effects: being geographic databases VLDBs (Very Large DataBases), updating instances can in-

Di Martino F., Sessa S., Polese G. and Vacca M. (2009).
A MAPREDUCE FRAMEWORK FOR CHANGE PROPAGATION IN GEOGRAPHIC DATABASES.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Databases and Information Systems Integration*, pages 31-36
DOI: 10.5220/0001861300310036

volve the moving of an enormous mass of data among geographical distributed nodes and their processing, making the process of propagating changes to the instances very expensive. As a consequence, db administrators have to cope with the efficiency of the change propagation process. The design of an evolution language is the first step toward the realization of a schema evolution tool and, according to Lagorce et al. (Lagorce et al., 1997), the instance update language (providing a set of instructions to update data after a change to the schema occurred), is one of its basic components. There are different practical update languages (Ferrandina et al., 1995; Lerner, 2000), but they do not allow the parallel processing of distributed data. In this paper we address the problem of designing an instance update language suitable for distributed databases and aiming, on the one hand, to widen the range of schema changes and, on the other hand, to make both easier and more efficient the work of spatial dataset administrators. We propose an instance update language based on the Map-Reduce-Merge paradigm (chih Yang et al., 2007), a recent extension of the MapReduce framework (Dean and Ghemawat, 2004) for heterogeneous datasets. MapReduce is the Google programming framework, which, for its efficiency, has become more and more popular, like it is proved by the many attempts to develop MapReduce based frameworks in many fields (chih Yang et al., 2007; Koufakou et al., 2007). The supporting execution model is mostly inherited from (Dean and Ghemawat, 2004; chih Yang et al., 2007) and we have been implementing a software system. The main results are the possibility of updating instances for a wide category of schema changes and of making this in an efficient way. The remaining of the paper is organized as follows: after a short introduction to spatial datasets (section 2), in section 3 the problem of change propagation is introduced and its features for spatial datasets are discussed. In section 4, after introducing the MapReduce frameworks, it is shown how a version of Map-Reduce-Merge can be used for change propagation in spatial datasets.

2 SPATIAL DATA SETS: BACKGROUND KNOWLEDGE

Spatial datasets are used to store a representation of geographical objects. In this work we use the ESRI UML geodatabase model schema (Zeiler, 1999; Arctur and Zeiler, 2004; Twumasi, 2002), an UML class-diagram representation used to design a conceptual model of a spatial dataset called *geodatabase-model*. An example of ESRI UML schema of a spatial database, representing routes of buses on a road network, is shown in fig. 1. The basic elements of the

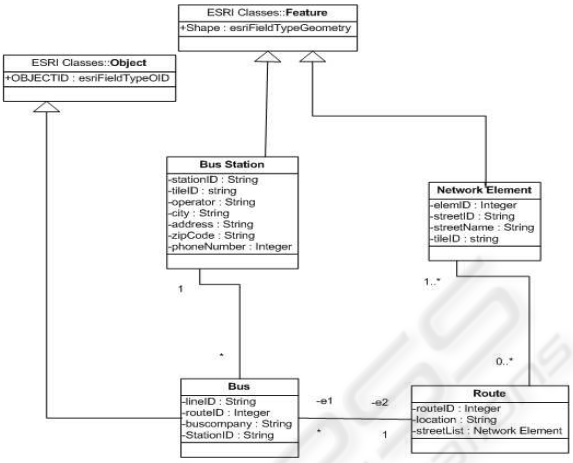

Figure 1: The **Bus Services**: a simple spatial dataset schema using the ESRI geodatabase model notation.

ESRI geodatabase model notation are: *feature classes* (collections of geographic features with the same geometry type, the same attributes, and the same spatial reference, like **NetworkElement** and the **BusStation**), *object classes* (classes of non spatial objects - like the class **Bus**), *relationship classes* (represents an association between feature classes or object classes - like the **RouteRel** relation between the classes **Route** and **NetworkElement**). There are many ways to store the data of a spatial dataset, depending on the specific spatial applications (Yeung and Hall, 2007). Many authors have experimented optimal distributed spatial data models related to typical spatial applications (Balovnev et al., 2000; Gorawski and Malczok, 2003; Yuhong and Jun, 2007). We suppose that the database space covers a wide area, which is divided into tiles, rectangular polygons that defines a rectangular coverage of the analyzed geographical area (Malinowski and Zimányi, 2006). The architecture we considers is a *multiple client/single server architecture* (Özsu, 2003) where to each client *node* is associated a set of contiguous tiles (see fig. 2). The data of the database (i.e. the features) are spread across the nodes using the horizontal fragmentation (Özsu, 2003), according to some spatial criteria.

3 SPATIAL DATA SETS: EVOLUTION

The problems of *semantics of changes* and the *change propagation* are very important in the context of schema evolution (Ram and Shankaranarayanan,

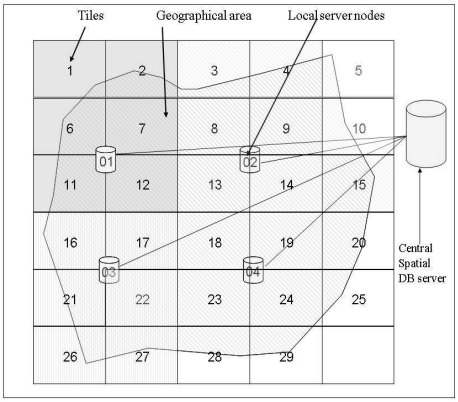

Figure 2: The multiple client/single server architecture of a spatial database.

2003). The first one refers to the way the changes are performed and their effect on the schema itself, while the second one deals with the effects on the data instances. Considering the schema in fig. 1, an example of simple change is the addition of the new string field *busType* to the class **Bus**. Changes to the schema can be *simple*, like the addition of a class or a field (see, for example, (Banerjee et al., 1987) for a taxonomy of simple schema changes) and *compound*, like merging two relations, which are very important in practical contexts (Brèche, 1996). There are different methods to realize the propagation of schema changes to the instances (see, for example, (Lerner, 2000)); in this paper we are interested in the *conversion method*, where a schema change provokes the update of all the objects affected by the change itself. *Instance-update languages* (Lagorce et al., 1997) serve to write programs to update instances after a schema change. Two notable examples of instance-update languages are those used in the O_2 system (Zicari, 1991) and in the TESS system (Lerner, 2000).

The ESRI package ArcGIS[1] is the most popular tool GIS in the world and it is equipped with many tools, among which there is ArcCatalog, which can be used to change the geodatabase schema. The main supported schema changes are: adding, modify or deleting attributes in a feature class or a class; adding new feature classes or classes; adding, modify or deleting domains; deleting relationship classes. One of the problem of this tool is its limitation, due to the small set of schema changes supported.

It is easy to maintain through examples (which we don't list here for brevity) that schema changes in spatial data sets have features which are not supported by the ArcGIS tools. In fact, first, the following observation about the kind of changes can be made: interesting schema changes for spatial datasets schema evolution are both simple and compound. According to Özsu (Özsu, 2003), the distribution of data, the communication and the processing costs are the main factors influencing the efficiency of query processing in distributed databases. As schema changes can be performed by queries (Bertino, 1992), these factors also influence the change propagation process: therefore, we analyse the complexity of the process of change propagation in their light and in that of the needed operations to process data due to the schema changes. For instance, changing an attribute type, with the subsequent conversion of the data from a type to another one, could be performed efficiently, if the data processing were made locally by the nodes. In fact, in order to modify the type of the attribute phoneNumber in the feature class BusStation from integer to string, the central server only needs to communicate the schema change to perform to the local nodes (which store the features of BusStation): each local node could operate the change on its own data and, at the end of the process, it could communicate either the operation success or the failure to the central server. Unfortunately, not all changes are so simple to be performed independently by each local node, for the need to perform operations involving data stored across more nodes (like the computation of the average length of a route). The previous considerations lead to the following observation about the cost: in order to minimize the change propagation cost in spatial datasets, it can be helpful to provide the instance update mechanism with two kinds of processing: local processing (in which an operation is executed on single nodes using only data of the single nodes themselves) and distributed processing (in which an operation, involving distributed data, is parallelized). The database administrator decides which kind of processing to use and when.

4 THE LANGUAGE FOR CHANGE PROPAGATION

4.1 MapReduce Frameworks

MapReduce is a programming model (Dean and Ghemawat, 2004) originally developed by Google to support parallel computations over vast amounts of data on large clusters of machines. The MapReduce framework is based on the two user defined functions *map* and *reduce* and its programming model is composed of many small computations using these two functions. In general, the *MapReduce execution pro-*

[1] http://www.esri.com/software/arcgis

cess (see (Dean and Ghemawat, 2004) for details) considers one of the copies of the user program calling map-reduce functions as special (it is called the master), while the rest are workers (there are M mappers and R reducers) the master assigns work to.

The MapReduce programming framework was born in the context of data retrieval, but its efficiency has made it more and more used in various applications, where distributed data and parallel processing are necessary (Chu et al., 2006; Koufakou et al., 2007; chih Yang et al., 2007). In this paper we are interested in the extension of the MapReduce to the case of heterogeneous data by the Map-Reduce-Merge model (chih Yang et al., 2007). This framework is based on three user defined functions (map, reduce and merge) with the following signatures (see (chih Yang et al., 2007) for details):

$$map : (k1, v1)_\alpha \rightarrow [(k2, v2)]_\alpha \quad (1)$$

$$reduce : (k2, [v2])_\alpha \rightarrow (k2, [v3])_\alpha \quad (2)$$

$$merge : ((k2, [v3])_\alpha, (k3, [v4])_\beta) \rightarrow [(k4, v5)]_\gamma \quad (3)$$

The meaning of each function is: a call to a map function processes a key/value pair $(k1, v1)$ and it returns a list of intermediate key/value pairs $[(k2, v2)]$; a call to a reduce function aggregates the list of values $[v2]$ with key $k2$ returning a list of values $[v3]$ always with the same key; a call to a merge function, using the keys $k2$ and $k3$, combines them into a list of key/value $[(k4, v5)]$ (see the subsection 4.2 for programming examples). Notice that a merge is executed on the two intermediate outputs $((k2, [v3])$ and $(k3, [v4]))$ produced by two map-reduce executions.

4.2 Map-Reduce-Merge for Change Propagation

The use of the Map-Reduce-Merge programming framework as the core of an instance update language has its theoretical justification in the fact that, being able to perform relational algebra operations on distributed data (chih Yang et al., 2007), Map-Reduce-Merge allows to ask queries, and, as a consequence (Bertino, 1992), it results also adequate to simulate changes. In order to use the Map-Reduce-Merge framework as an instance update language exploiting the features of the evolution of spatial data sets seen in the section 3, it is necessary to enrich it with:

- the possibility of choosing the processing mode;
  This is made introducing the instruction of processing mode selection:
  $$processing\ mode = local|distributed$$
- the possibility of explicitly referring to the tables to be used.
  This is made introducing the instruction:
  $$use\ input|output\ \langle table\ name \rangle$$
  The *input* parameter serves the programmer to specify the classes to be used (and the master to locate the node of the mappers - see afterwards); the *output* parameter of the instruction is used to combine the data in the specified table.

Moreover, as the instance update language has to have fragmentation capability (see sect. 2), our framework also provides the instruction:

$$divide\ \langle table\ name \rangle$$

Consider the schema in figure 1, the following simple example shows the use of map and reduce to compute the total number of elements for each route. The result is the table of route length (*routeID, RouteLenght*). Notice that, in map, reduce, and merge functions, users have to group the attributes to be used in two subsets, the key part and the value part (for instance, routeID and elemID are, respectively, the key part and the value part in the next map function).

```
processing mode = distributed;
use input RouteRel;
map(const Key& key,
    const Value&, value){
    routeID = key;
    elemID  = value;
 output_key = (routeID);
 output_value = (elemID);
 Emit(output_key, output_value);
/*This map retrieves the pairs (routeID,elemID),
  and buffers them ordered by routeID.
  There is no user function.*/
}
sort by routeID; /*see afterwards*/
reduce(const Key& key,
       const ValueIterator& value){
 int RouteLenght = 0;
 for each (k in key)
       RouteLenght += 1;
 Emit(key, (RouteLenght);
/*This reduce, using an iterator, reads the
buffered pairs (routeID,elemID) and, for each
unique intermediate key (routeID), it computes
the length of the route.*/
}
```

In the model we propose, changes to the geodatabase schema are performed only by the central server. Like in (Dean and Ghemawat, 2004), a program with map-reduce-merge calls, when executed, generates copies of itself. One of the copies is the *master* (also coordinator in (chih Yang et al., 2007)), the other ones are the *workers*. The master coordinates all the processing operations and, therefore, it also manages the processing modes. Afterwards,

the execution process is illustrated, which is inherited from (Dean and Ghemawat, 2004; chih Yang et al., 2007), except for some minor differences due to the presence of the local mode and the management of the merge iterators.

- *Map task*
  When a map is encountered, the master assigns the map tasks to the M workers (mappers). In the previous example, the master, using the fragmentation information, locates the nodes storing the **RouteRel** data and on them launches the mappers. A map task consists in reading data from the input locations, passing them to the user map function and, then, storing them, sorted by the output key, at some locations on some nodes. In the previous example, map tasks store the intermediate data at some locations according to some criterion on the key *routeID*. There is no difference between local and distributed mode for the map task.
- *Reduce task*
  The master passes the locations where the mappers have stored the intermediate data to the R reduce workers (reducers) which are assigned to some nodes. The reducers, using an iterator, for each unique intermediate key (e.g. each value of *routeID*), pass both the key itself and the corresponding list of values (e.g. the list of *elemID*) to the users reduce function (e.g. the computation of the length). The result of the user reduce function is stored on some nodes. In local mode, the number of reducers is equal to the number of mappers, as they are located on the same nodes of the mappers.
- *Merge task*
  When the user program contains a merge call, the master launches the merge workers (mergers) on a cluster of nodes. In (chih Yang et al., 2007), mergers take data, to be passed to the user merge function, from two sources (the locations where reducers stored them) using both a partition selector and a configurable iterator. In order to avoid the users to manage iterators in some cases, we introduce the instructions *sort by* $\langle names \rangle$ and *match* $\langle name_1 \rangle$ *with* $\langle name_2 \rangle$. The first one sorts the reduce output by the keys indicated by *names*, while the second one guarantees that the output of the two reduce functions is stored in buckets using the same range of values (matchable values). In local mode, the merge function is executed on the local nodes involved in the processing and on the related reduce locations.

To continue the previous example, using the merge function, it is possible to perform the addition of the new field *RouteLength* in the class **Route**:

```
use input route;
map(const Key& key,
    const Value&, value){
    routeID = key;
    location = value.location;
    Emit((routeID),(location));
}
sort by routeID;
match routeID with routeID;
reduce(const Key& key,
       const Value& value){
 Emit(key, value}
}
merge(const LeftKey& leftKey,
      const LeftValue& leftValue,
      const RightKey& rightKey,
      const RightValue& rightValue)
 if (leftKey == rightKey){
 Emit(leftKey,rightKey);}
 /*The merge joins on routeID.*/
use output route; /*The class route is*/
divide route;  /*fragmented.*/
}
```

Considering the schema in fig. 1, the following example shows how to perform a compound change: the creation of the new class **NewBus** obtained projecting the class **Bus** on *routeID* and *buscompany*, projecting the class **Route** on *routeID* and *location*, and, finally, merging the results on the attribute *routeID*.

```
processing mode = distributed;
use input Bus;
map(const Key& key,
    const Value&, value){
    routeID = key;
    buscompany = value.buscompany;
 output_key = (routeID);
 output_value = (buscompany);
 Emit(output_key, output_value);
/*The map emits pairs routeID and buscompany.*/
}
sort by routeID;
reduce(const Key& key,
       const Value& value){
 Emit(key, value}
}
use input route;
map(const Key& key,
    const Value&, value){
    routeID = key;
    location = value.location;
 Emit((routeID),(location));
 /*The map emits pairs routeID and location.*/
}
sort by routeID;
match routeID with routeID;
reduce(const Key& key,
       const Value& value){
 Emit(key, value}
```

```
}
merge(const LeftKey& leftKey,
      const LeftValue& leftValue,
      const RightKey& rightKey,
      const RightValue& rightValue)
 if (leftKey == rightKey){
 Emit(leftKey,RightKey);}
 /*The merge joins on routeID.*/
}
use output NewBus;/*The class NewBus is*/
divide NewBUs; /*fragmented.*/
```

5 CONCLUSIONS AND FURTHER WORK

A language for change propagation in spatial datasets, extending the capability of existing tools, has been presented. The system (under development) uses ArcGIS 9.3 software, the ESRI SDE package, the ESRI ArcObjects on Java platform and the RDBMS ORACLE 10g. We have also planned to design and implement both a visual language for schema changes and the mechanism for the semi/automatic generation of map-reduce-merge routines to propagate changes.

ACKNOWLEDGEMENTS

The authors would like to thank Vittorio Scarano (University of Salerno) for his contribution to the birth of this work.

REFERENCES

Arctur, D. and Zeiler, M. (2004). *Designing Geodatabases: Case Studies in GIS Data Modeling*. ESRI Press, Redlands (CA).

Balovnev, O. T., Breunig, M., Cremers, A. B., and Shumilov, S. S. (2000). Extending geotoolkit to access distributed spatial data and operations. In *SSDBM*, pages 259–261.

Banerjee, J., Kim, W., Kim, H.-J., and Korth, H. F. (1987). Semantics and implementation of schema evolution in object-oriented databases. In Dayal, U. and Traiger, I. L., editors, *SIGMOD Conference*, pages 311–322. ACM Press.

Bertino, E. (1992). A view mechanism for object-oriented databases. In Pirotte, A., Delobel, C., and Gottlob, G., editors, *EDBT*, volume 580 of *Lecture Notes in Computer Science*, pages 136–151. Springer.

Brèche, P. (1996). Advanced principles for changing schemas of object databases. In Constantopoulos, P., Mylopoulos, J., and Vassiliou, Y., editors, *CAiSE*, volume 1080 of *Lecture Notes in Computer Science*, pages 476–495. Springer.

chih Yang, H., Dasdan, A., Hsiao, R.-L., and Jr., D. S. P. (2007). Map-reduce-merge: simplified relational data processing on large clusters. In Chan, C. Y., Ooi, B. C., and Zhou, A., editors, *SIGMOD Conference*, pages 1029–1040. ACM.

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G. R., Ng, A. Y., and Olukotun, K. (2006). Map-reduce for machine learning on multicore. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *NIPS*, pages 281–288. MIT Press.

Dean, J. and Ghemawat, S. (2004). Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150.

Ferrandina, F., Meyer, T., Zicari, R., Ferran, G., and Madec, J. (1995). Schema and database evolution in the o2 object database system. In Dayal, U., Gray, P. M. D., and Nishio, S., editors, *VLDB*, pages 170–181. Morgan Kaufmann.

Gorawski, M. and Malczok, R. (2003). Distributed spatial data warehouse. In Wyrzykowski, R., Dongarra, J., Paprzycki, M., and Wasniewski, J., editors, *PPAM*, volume 3019 of *Lecture Notes in Computer Science*, pages 676–681. Springer.

Koufakou, A., Ortiz, E. G., Georgiopoulos, M., Anagnostopoulos, G. C., and Reynolds, K. M. (2007). A scalable and efficient outlier detection strategy for categorical data. In *ICTAI (2)*, pages 210–217. IEEE Computer Society.

Lagorce, J.-B., Stockus, A., and Waller, E. (1997). Object-oriented database evolution. In Afrati, F. N. and Kolaitis, P. G., editors, *ICDT*, volume 1186 of *Lecture Notes in Computer Science*, pages 379–393. Springer.

Lerner, B. S. (2000). A model for compound type changes encountered in schema evolution. *ACM Trans. Database Syst.*, 25(1):83–127.

Malinowski, E. and Zimányi, E. (2006). Requirements specification and conceptual modeling for spatial data warehouses. In Meersman, R., Tari, Z., and Herrero, P., editors, *OTM Workshops (2)*, volume 4278 of *Lecture Notes in Computer Science*, pages 1616–1625. Springer.

Özsu, M. T. (2003). Distributed database systems. In *Encyclopedia of information systems*, pages 673–682.

Ram, S. and Shankaranarayanan, G. (2003). Research issues in database schema evolution: the road not taken. *Univ. of Arizona, Working Paper #2003-15*.

Twumasi, B. O. (2002). Modelling spatial object behaviours in object-relational geodatabase. *Int. Inst. for Geo-inf. Science and Earth Observ. (ITC), Master thesis*.

Yeung, A. K. W. and Hall, G. B. (2007). *Spatial Database Systems: Design, Implementation and Project Management (GeoJournal Library)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Yuhong, W. and Jun, C. (2007). Generic framework and key issues for updates propagation between heterogeneous spatial databases. In *ISPRS Workshop on Updating Geo-spatial Databases with Imagery & The 5th ISPRS Workshop on DMGIS*, pages 215–221.

Zeiler, M. (1999). *Modeling Our World: The ESRI Guide to Geodatabase Design*. ESRI Press, Redlands (CA).

Zicari, R. (1991). A framework for schema updates in an object-oriented database system. In *ICDE*, pages 2–13. IEEE Computer Society.