Problems during Database Migration to the Cloud

Goran Vodomin, Darko Andročec
Faculty of Organization and Informatics
University of Zagreb
Pavlinska 2, 42000 Varaždin, Croatia
{gvodomin, dandrocec}@foi.hr

Abstract. *Numerous heterogeneities among different local databases and cloud storages make database migration to the cloud an interesting and complex research and practical problem. Successful execution of more complex interoperability scenarios cannot be imagined without being able to move data from local servers to the cloud, and from one cloud storage to another. In this work, the authors have developed a working prototype of a migration tool for MySQL, PostgreSQL and Microsoft SQL Server. The main contribution of this work is the identification of problems that can occur during database migration process to the cloud. The authors also list differences between data storage models of various commercial cloud providers to envisage possible issues when moving data from one cloud storage to another.*

Keywords: Cloud, data migration, migration tool, migration problems

1 Introduction

Due to its advantages, cloud computing is now used by many business and public sector organizations. The most common definition of cloud computing is the National Institute of Standards and Technology's (NIST's) formulation, claiming that this paradigm "is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, services) that can be rapidly provisioned and released with minimal management effort or service provider interaction" [4]. Every computing paradigm has some limitations. One of the main obstacles of cloud computing is provider data lock-in. This problem is characterized by complex and time-consuming migration of data to alternative cloud vendors. There are also problems when trying to initially migrate data from local servers to the cloud. The main aim of this paper is to identify issues that can occur during database migration process to the cloud. The authors also list differences between data storage models of various commercial cloud providers to envisage possible problems when moving data from one cloud storage to another.

This paper proceeds as follows. First, in Section 2, the related work is listed. In Section 3, data migration to the cloud is explained in more detail. Section 4 shows sample database structure used for testing the authors' migration tool. In Section 5, the authors' data migration tool is presented together with its main functionalities. Section 6 and Section 7 present the identified migration problems and differences among cloud storages. Conclusions are provided in the final section.

2 Related work

There are some existing works dealing with data migration from the existing systems to the clouds, or from one cloud offer to another. Ranabahu and Sheth [6] present the usage of semantic technologies to overcome cloud vendor lock-in issues. They distinguish four types of semantics for an application: data semantics (definitions of data structures, their relationships and restrictions), logic and process semantics (business logic of the application), non-functional semantics (e.g. access control and logging) and system semantics (deployment descriptions and dependency management of the application).

Miranda et al. [5] used software adaptation techniques to tackle cloud interoperability and migration. Software adaptation techniques are aimed at developing mediator elements, called adaptors. They identified three important interoperability problems of cloud service based applications: communication is conditioned by the technology supported by each vendor, invoking third-party services is limited by the supported invocation mechanisms, and portability problems occur due to vendor-specific technologies. The variability among different providers' APIs and service specifications can be defined by using formal methods and by generating the required mappings and adaptation components. Bastiao Silva et al. [1] developed a unified API for delivering services using cloud resources of multiple vendors with abstract layer for cloud blob stores, cloud columnar data (e.g. *Azure*

Table), and Publish/Subscribe mechanism (*Channel API* of *Google App Engine* and *Azure Queue*).

There are several cloud APIs and frameworks that act as intermediaries between different clouds. *Apache Libcloud* is a Python library containing a unified API that can manage cloud resources of different providers. This library is focused on infrastructure as a service and supports cloud servers, block storage, cloud object storage, load balancers, and Domain Name System (DNS) as a service. *Deltacloud* API contains a cloud abstraction API working as a wrapper around a large number of clouds to abstract their differences. It is also focused on infrastructure as a service (IaaS) providers and provides drivers for Amazon, Eucalyptus, GoGrid, OpenNebula etc. *Apache jclouds* is an open-source library offering blob (binary content) store and compute service abstraction for 30 IaaS providers. There are also some commercial (industrial) approaches to tackle cloud portability and interoperability. For example, *Cloutex* can integrate and synchronize data between Salesforce, Quickbooks Online and Magento. A similar offer, *Import2.com*, enables the transfer of data between cloud application such as Salesforce, Tumblr, Nimble, Pipedrive, SugarCRM, and Zoho Customer Relationship management (CRM) software. *Import2* is currently focused on CRM, helpdesk and blog migration of cloud data. The two offers mentioned here are focused on software as a service (SaaS) data.

3 Data migration to the cloud

Data migration to the cloud is the process of moving data from the existing databases to the databases that are placed in the cloud. During this process, data should not be lost or modified. There are many reasons for moving data to the cloud and some of them are: system upgrade, data consolidation, improving data security, etc. [7]. Information systems often have different models of data storages, so data often needs to be converted from one format to another. Through the conversion process, companies often enhance the model of storage and some of the data is deleted or modified. After the migration process is done, organizational structure is often changed and it is necessary to prepare staff to work with the new information system. Each data migration has specific needs and most data migrations differ, because their characteristics depend on the systems from which and to which data is migrated.

The process of migration is not easy and needs to be well prepared and organized. That is why the migration can be divided into several steps. According to the authors ([3], [8]) data migration consists of eight steps:

1. Define the scope of migration
2. Ensure data security
3. Select service provider
4. Mapping the data
5. Scheduling the migration
6. Select tools for migration or develop migration scripts
7. Testing before and after the migration.
8. Actual data migration

The amount of time that will be required to migrate relational database depends on the amount of data that is stored in the database, the number of stored procedures, views and triggers. Data definition language (DDL) migration can last between a couple of days or two weeks, during which testing must be included. There are many tools for relational database migration such as *MySQL Workbench*, *SQL Server Management Studio*, *MicroOLAP DB Designer* etc. Regardless of the variety of tools that simplify the migration, there are some changes in the database that the tool will not recognize (table and column splitting, data type or table name changed). Relational database migration can be divided into three major phases [3]:

1. Relational schema migration – it includes the migration of tables, indexes and views.
2. Data migration done via tools or migration scripts. The time required for data migration depends on the size of the database.
3. Database stored programs migration – the migration of stored procedures and triggers.

4 Test database

To simulate and test the migration process, the authors of this study developed *Transport* database. The database shows a transportation company's data. The most important information is stored within the tables *vehicle_work* and *transportation_order*. Table *vehicle_work* contains data about business partner, vehicle, employee etc. and table *transportation_order* contains data about products which need to be transported. Besides data, the database also contains stored procedures (*verify_date, verify_cargo*), triggers (*trig_date_check*, *tirg_cargo_check*) and views (*vehicle_list*, *employee_list*, *warehouse_list*). Entity-relationship model of "Transport" database is developed in *MySQL Workbench 6.0* and shown in Figure 1.

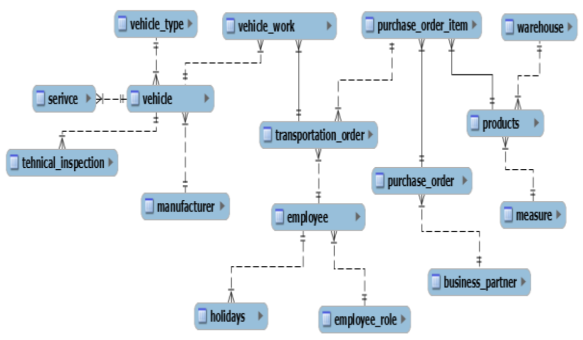


Figure 1. Transport entity-relationship model

5 Data migration tool

There are several ways (tools, migration scripts) to migrate relational database to the cloud. Migration scripts are often used for modifying and database administration. Migration script can include changes like database schema modification, data modification, database management systems (DBMS) upgrade, user management etc. [2]. Database administrators like to develop their own scripts for database maintenance, however, if the administrator leaves the organization, new administrators do not have full knowledge of how to properly use the script. There are two kinds of migration scripts:

1. Automated migration script – generated by the software for data migration, example MySQL Workbench
2. Manual migration script – developed by database programmer.

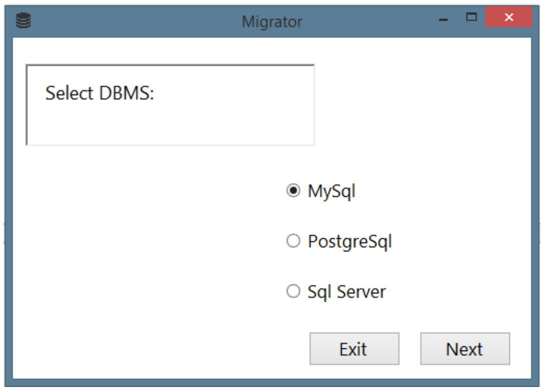
Migration tools can be open source or commercial. For the purpose of this study, a simple data migration tool was developed. The tool name is *Migrator* and it is developed using Microsoft .Net technology. It is aimed at migrating data from local database to cloud storages. *Migrator* can also be used for some kind of cloud-to-cloud migration. In that case, the data from cloud provider is stored on the local storage and it is used for the migration process on another cloud provider. The time needed for the migration heavily depends on the speed of the internet connection, because program stores data on the local storage and then restores it on another cloud provider.

In the "real world" there are numerous types of relational database management systems (RDBMS) and *Migrator* can, for now, work with the following three popular DBMS systems: MySQL, PostgreSQL and Microsoft SQL Server. For each type of RDBMS there is a different set of libraries and methods to work with. Commercial tools can migrate relational database from one RDBMS to another RDBMS, but in most cases they are expensive. There are also open source tools and scripts which can be used for migration from one RDBMS to another RDBMS, but the problem is that these tools and scripts work only with some RDBMS systems and often cannot migrate stored procedures, triggers and views. *Migrator* is a free migration tool, still in development, which works under Windows platform. The benefits of its use are: it is easy to use, does not require a lot of computing resources, creates a data backup, it can be used for cloud-to-cloud migrations, it has the ability to migrate only selected database objects, etc. Although other free and commercial tools for data migration exist, it is difficult to determine the most common interoperability problems and how to resolve them without developing a new tool. This is the main reason why the authors decided to develop a new tool. *Migrator* can be downloaded from *GitHub* public repository:
https://github.com/GorskiV/Migrator.

If the migration must be carried out between different RDBMS systems, data conversion must be made. Each RDBMS has its own syntax, datatypes and a way of writing stored views, procedures and triggers. Because of the mentioned problems, *Migrator* does not support the migration between different RDBMS systems for now. On the other hand, commercial tools (for example *SwisSQL, Oracle SQL Developer, ESF Database Migration Toolkit)* can overcome the afore mentioned problems and these functionalities make the difference compared to *Migrator*. In addition, commercial tools can work with more RDBMS types.

Migrator consists of seven main classes that are necessary to complete the migration of relational database. The most important class is *MigratorModel* which is aimed at working with three different types of RDBMS. Within this class, methods are used to connect, retrieve, backup and restore data from different RDBMS systems. Methods which are used for establishing connection are *MySqlConnect*(), *SqlServerConnect*() and *PostgreSQLConnect*(). Each of these methods has a connection string as an argument which is provided by the user. After establishing the connection, based on which RDBMS is chosen, different sets of methods are used. For example, if the user chooses MySQL for retrieving data, the program uses the following methods: *MySqlTables*(), *MySqlViews*(), *MySqlProcedures*() and *MySqlTriggers*(). The same principle is applied for retrieving data when PostgreSQL and *SQL Server* are selected. Each method uses a connection string as an argument to connect to a database. Things get more complicated when backup and restore have to be taken into consideration. For backing up and restoring *PostgreSQL* database, *Migrator* uses third party software *pg_dump* and *pg_restore*. Before backing up and restoring *PostgreSQL* database program verifies if this software is installed on user computer. Methods used for verifying are *verifyPostgreInstalation*(), PG*_DumpExePath*(), *LookForFile*(), *performFileSearchTask*() and they are developed by Vinay Swa. After data is retrieved, the user selects which data will be migrated. The user chooses tables, views, triggers and procedures for the migration process. Depending on the selected data, different methods are called for data backup. Arguments for backup methods are RDBMS type, connection string, backup path and data list. When *SQL Server* is selected, the user does not have an option for choosing the data that will be migrated. The problem is that there are not methods and libraries available for backing up specified data. In this case, the user can backup and restore the entire database. In the end, previously backed data is used for restoring on cloud provider. There are three methods for restoring data on cloud provider: *MySqlImport*(), *PostgreSQLBackupImport*() and *SqlServerImport*(). The most important arguments for restoring data are connection string and path to the previously backed data. The migration process and methods for restoring data are called when the user presses the button *Start Migration*. It is important to note that external libraries and third party software are used for the entire migration process.

Figure 2 shows the main form. The form displays three different options for choosing RDBMS. Based on the selection, different sets of methods are used within *MigratorModel* class during the process of migration. For example, if the user chooses *MySQL* in the further process it is necessary to provide data (connection strings) which will work with this type of RDBMS. When selecting *PostgreSQL* RDBMS *Migrator* checks if there are *pg_dump* and *pg_restore* tools installed on user system. These are external tools which are needed for backup and restoring of data from *PostgreSQL* database. Microsoft .Net technology provides *Npgsql* library to work with *PostgreSQL* but within *Npgsql* library there are only methods and classes which select and insert data from database, they do not support backup and restore data.

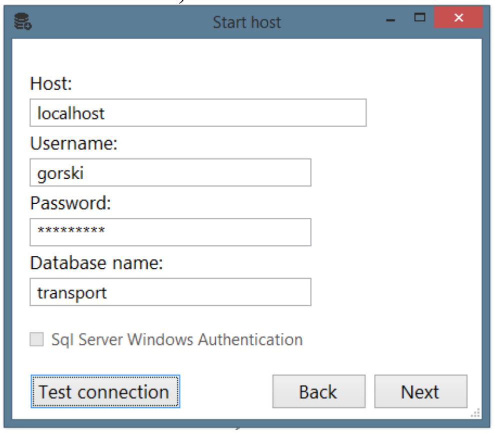

Migrator

Select DBMS:

[✓ MySql] PostgreSql Sql Server

Figure 2. Main form of the *Migrator* tool

Figure 3 shows the form for entering connection parameters for the chosen RDBMS. Once the user enters data, he can test the connection, and if the data is valid, the connection is successful. Checkbox *SQL Server Windows authentication* is checked if the user intends to connect on SQL Server which requires Windows authentication (in most cases this is local *SQL Server* database).


Start host

Host: localhost

Username: gorski

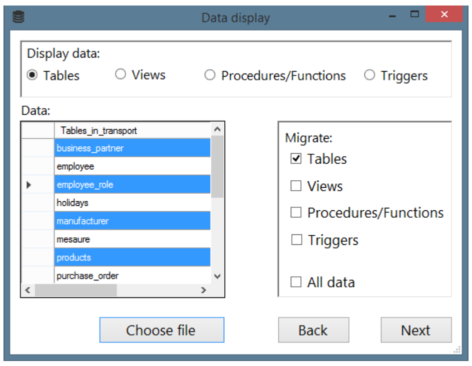
Password: *********

Database name: transport

☐ Sql Server Windows Authentication

Figure 3. Entering connection parameters

After the user connects to database, Figure 3 displays all the data within database. This data can be tables, views, procedures and triggers. The form header displays options for choosing which data will be displayed inside the table. On the right side there are also options for selecting the data that will be migrated. The user does not have to choose all tables, views or procedures for the migration process. He can select only a few tables or views which will be migrated. By clicking on the *Next* button, program backups all the selected data. This backup will be used in the further process of the migration.


Data display

Display data: [✓ Tables] Views Procedures/Functions Triggers

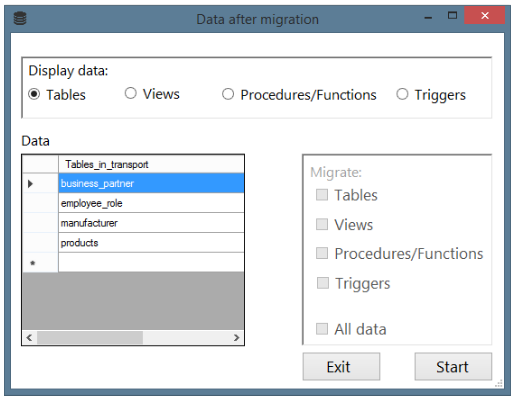
Data:

Tables_in_transport
business_partner
employee
employee_role
holidays
manufacturer
mesaure
products
purchase_order

Migrate: ☑ Tables ☐ Views ☐ Procedures/Functions ☐ Triggers ☐ All data

Figure 4. Select data for migration

When backup is completed, a new form is opened similar to Figure 3. On this form the user provides connection parameters for the selected target database. Database can be located on localhost (for testing purposes) or on the cloud. There is also *Test connection* button where the user can check if the connection parameters are valid. If the connection is successful, the user can start the migration. Migration data is retrieved from the previously created backup. If all goes well, the program informs the user about successful migration. On the other hand, if there is a problem during the migration, the program raises error notice and aborts the migration. In the end, after the migration is completed, GUI shows the migrated data on cloud database, as seen in Figure 5, and the user can start a new migration or close the program.


Data after migration

Display data: [✓ Tables] Views Procedures/Functions Triggers

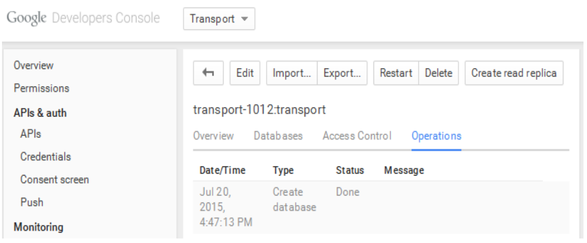
Data

Tables_in_transport
business_partner
employee_role
manufacturer
products

Migrate: ☐ Tables ☐ Views ☐ Procedures/Functions ☐ Triggers ☐ All data

Figure 5. Migrated data

Data migration is carried out on the Google Cloud platform and all the data is migrated (tables, views, procedures, triggers). Figure 6 displays the created database on Google Cloud Platform.



transport-1012:transport

Date/Time	Type	Status	Message
Jul 20, 2015, 4:47:13 PM	Create database	Done	

Figure 6. Google Cloud SQL

6 Problems identified during migration

During relational database migration various problems may occur. Most of the problems depend on the chosen cloud service provider. It is often necessary to make server configuration before the migration step. Not all cloud providers will provide super roles during database and server configuration and this can be a problem. In some relational database management systems, functions, procedures and especially triggers need to have super role to be able to work. In some cases, data conversion is required, if the migration has to be completed between different RDBMS systems. For each cloud provider, there are some problems which occurred during the migration. Problems are related with server configuration and the migration of stored procedures, triggers and functions.

For example, Amazon AWS has the following requirements when working with their MySQL instances:

- Data type configuration – date format has to be *year-month-day-hour-min-sec*. During the migration of date data (date data is contained in the following tables of the authors' sample database: *vehicle_work*, *holidays*, *purchase_order* etc.) schema migration of mentioned tables preformed successfully, but if the date data was not in the correct format, it will not migrate. To overcome these problems during data backup, date format has to be specified. Date format has to correspond with the format of cloud database, and then date data migration performed well.
- Server configuration – before the migration of stored procedures, trigger and functions parameter *log_bin_trust_function_creators* has to be set on 1
- Data configuration – user defined functions have to contain clause *DETERMINISTIC,* or parameter *log_bin_trust_function_creators* has to be set on 1. During the migration of stored procedures (*verify_date*, *verify_cargo*) and triggers (*trig_date_check*, *tirg_cargo_check*) clause *DETERMINISTIC* was not included because server parameter *log_bin_trust_function_creators* was set on 1.
- Server configuration – if user defined functions, views, stored procedures and triggers start with uppercase, parameter *lower_case_table_names* has to be set on 2. Transport database does not contain objects that start with uppercase and therefore this parameter has not been changed.

Google Cloud SQL has only MySQL offer, there is no offer for PostgreSQL or SQL Server RDBMSs. Also, in its MySQL offer, there is a requirement that - for stored functions, views and triggers - the attribute DEFINER has to be excluded because it requires super privileges which are not provided. For example, if the attempt is made to migrate trigger with the following code: `create definer=gorski trigger trig_date_verification before insert on vehicle_work…` the migration will fail. While creating a backup, attribute DEFINER has to be excluded and later, after the migration is done, database administrator defines privileges (using GRANT statement) for using procedures, triggers etc... Microsoft Azure does not support neither MySQL nor PostgreSQL. Azure version of SQL Server RDBMS has the following additional requirements:

- There is a need for additional configuration for composite foreign keys. Transport database contains a table (purchase_order_item) which has composite foreign keys (fk_purchase_order and fk_products). Composite foreign keys require a clustered index, and after table schema migration, on older versions of Azure SQL Server, clustered indexes have to be created manually before data migration, otherwise data migration will fail. Azure SQL Database's latest update (V12) allows tables without clustered indexes, and if upgrade is done, there is no need for clustered indexes. In this case, the authors upgraded database on the latest update and after the upgrade the migration performed well.
- There is no support for triggers.

7 Differences between cloud storages

There are many data migration/interoperability problems among cloud storages. The first identified problem is the difference between data storage models. As an illustration, it is difficult or even impossible to move data without losing important information from an SQL model of one provider to a NoSQL model of another platform as a service provider. Even if the same models were chosen (e.g. SQL) in two various offers, these models will still have significant differences due to provider's design and used technology. For example, each provider supports their own set of data types. Data types differ in name, value space, permitted range of values, precision of data etc. Some offers also have predefined standard objects or tables. Data import or export is often complicated. Most providers offer only basic CSV or XML exports (list of columns and row data), so users cannot determine data types, identifiers, possible relationships between tables (e.g. foreign keys) etc. Users must use remote APIs of cloud providers to get that information. APIs are not standardized, so users need to cope with different functions, input and output parameters and different means to access remote API functionalities by using libraries for programming languages and/or SOAP or REST web services. Various platform as a service providers also use their own versions of data query languages.

8 Conclusion

There are many data migration problems when trying to move from local servers to the cloud, and when migrating data from one cloud offer to another. Some cloud vendors simply do not support some databases (for example, Google Cloud SQL has only *MySQL* offer, *Microsoft Azure* does not support neither *MySQL* nor *PostgreSQL*). Users are limited to choose provider that supports their preferred database, or they must use some migration scripts or tools to execute mappings and migrate to another cloud storage option.

To minimize the possible data migration problems, users should carefully choose cloud storage offer. It is also best to avoid using vendors' specific features that are not supported in any other cloud offer. For example, most data type problems can be avoided if the established variants of data types (for example, integer, string etc.) are used. The more users use advanced and innovative functionalities that are provider or cloud storage offer specific, the more difficult it will be for migration and interoperability to occur. The future work could include several improvements of this migration tool. For example, the authors could consider how to migrate data from relational databases to *NoSQL* cloud storages and how to perform data type mappings between them.

Acknowledgments

The authors would like to thank the reviewers for their comments that helped improve the manuscript. This work has been fully supported by the Croatian Science Foundation under the project IP-2014-09-3877.

References

[1] Bastião Silva, LA; Costa, C; Oliveira, JL. A common API for delivering services over multi-vendor cloud resources. *Journal of Systems and Software*. 86(9):2309–17, 2013

[2] Hickford, J. Using Migration Scripts in Database Deployments. https://www.simple-talk.com/sql/database-administration/using-migration-scripts-in-database-deployments/, downloaded: August 8th, 2014

[3] Laszewski, T; Prakash, N. (2011). Migrating to the Cloud. http://www.oracle.com/technetwork/articles/cloudcomp/migrating-to-the-cloud-chap-3-495856.pdf, downloaded: August 11th, 2014

[4] Linthicum D.S.: Cloud Computing and SOA Convergence in Your Enterprise, Addison-Wesley, New York, USA, 2009

[5] Miranda, J; Murillo, JM; Guillé, J; Canal, C. Identifying adaptation needs to avoid the vendor lock-in effect in the deployment of cloud SBAs. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups (WAS4FI-Mashups '12)*, Bertinoro, Italy, 2012

[6] Ranabahu, A; Sheth, A. Semantics Centric Solutions for Application and Data Portability in Cloud Computing. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom 2010)*, pages 234-241, Indianapolis, USA, 2010

[7] Russom, P. Best Practices in Data Migration, http://download.101com.com/pub/TDWI/Files/TDWI_Monograph_BPinDataMigration_April2006.pdf, downloaded: July 25th, 2014

[8] Swa, V. Postgres Database Backup/Restore From C#. http://www.codeproject.com/Articles/360472/Postgres-Database-Backup-Restore-From-Csharp, downloaded: May 5th, 2015