Подход к верификации корректности миграции данных между СУБД с использованием криптографических хэш-функций

Максим Журавлев, Виктор Полозов
Кафедра системного программирования,
Санкт-Петербургский государственный университет

24 сентября 2013 г.

Аннотация

В данной статье описывается автоматизированный подход к сравнению содержимого баз данных в промышленном проекте по миграции БД из СУБД MS SQL Server 2005 в Oracle 11gR2. Целевая БД должна содержать те же данные, что и исходная с точностью до ограничений, налагаемых используемыми СУБД, и быть функционально эквивалентной. Размер данных в исходной базе составляет порядка 6 терабайт, размер кода - 2.5 миллиона строк кода в хранимых процедурах. Разработан метод сравнения содержимого БД независимый от порядка просмотра записей, использующий коммутативные и криптографические хэш-функции. Показана применимость данного метода для практического использования.

Ключевые слова - базы данных, миграция данных, тестирование, реинжиниринг, обеспечение качества, хэширование

1. Введение

Перед коллективом, в котором имеют честь работать авторы, была поставлена задача произвести миграцию промышленной БД из Microsoft SQL Server 2005 в Oracle Database 11g Release 2.

Проект условно можно разделить на следующие части: миграцию кода хранимых процедур, представлений (VIEW) и триггеров, миграция данных и обеспечение качества.

При миграции производилась доработка кода с учётом специфики целевой СУБД, новых требований к системе, удалялся мёртвый код, и проводились другие согласованные с заказчиком изменения. При миграции схемы БД проводились такие изменения, как переименование процедур, таблиц и колонок, в основном связанные с ограничением СУБД Oracle в 30 байт на имя, и согласованные изменения в схеме, такие как разбиение по схемам и удаление не используемых объектов.

Миграция данных проводилась без наличия прямого подключения между базами, через временные файлы с использованием инструментов Microsoft

SQL Server BCP и Oracle SQL*Plus. Конвертация данных проводилась в три этапа: часть данных конвертировалась при выгрузке, часть при загрузке, оставшиеся конвертации проводились как отдельный шаг после загрузки.

В настоящей статье даётся более технически подробное описание реализованного авторами этапа тестирования – верификации корректности миграции путем сравнения набора хэшей.

2. Связанные работы

Методологии миграции баз данных описаны в нескольких статьях. В статье [7] представлен пример методологии миграции данных. Процесс миграции между базами данных с отличающимися моделями данных, сопутствующие риски и проблемы описаны в [9] и [1]. Методологии миграции legacy-систем представлены в [15].

Проекты миграции баз данных сталкиваются со специфичными рисками и проблемами. Типичные риски и подходы к тестированию и обеспечению качества описываются в [11]. Разновидности тестирования, применяемые в течение жизненного цикла проекта миграции, рассматриваются в [12]. Общая схема организации обеспечения качества нашего проекта описана в статье [8].

Вопросы хэширования неупорядоченных коллекций (множеств и мультимножеств) рассматриваются в [4]. В этой же работе впервые вводится определение устойчивости хэш-функции множества (мультимножества) к коллизиям. Хэширование неупорядоченных коллекций тесно связано с инкрементальным хэшированием. Результаты с использованием операции XOR получены в работах [10] и [3]. Подход к хэшированию таблиц, использованный в нашем проекте, напоминает *MSet-XOR-Hash* из статьи [4].

Схожие подходы используются в похожей задаче поддержания консистентности реплицированных баз данных. К таким работам относятся: статьи [5], [6] и доклад [2]. В данных статьях не рассматривается сравнение БД, находящихся под управлением разных СУБД.

3. Требования к методу

На этапе планирования проекта было решено, что совпадение данных в исходной и целевой БД должно быть проверено отдельным инструментом. Были сформулированы требования к инструменту:

1. Сравнение БД, находящихся в разных СУБД. Все используемые методы должны быть реализованы в MS SQL Server 2005 и Oracle 11gR2.

2. Настраиваемая процедура сравнения.

3. Независимость результата сравнения от порядка записей в выборках. Это требование связано с тем, что СУБД не гарантируют порядок выдачи результатов для выборки без явной директивы *ORDER BY*. От добавления явной директивы для упорядочивания отказались по двум причинам: отсутствие первичного ключа или уникального индекса у части таблиц, и существенное увеличение времени работы при упорядочивании без использования индекса.

4. Эффективность работы. Сюда включается возможность достаточно эффективной реализации и возможность распараллеливания в обеих СУБД. Приемлемым считалось время сравнимое со временем работы процедуры перегрузки данных.

От требования определения места расхождения было решено отказаться, т.к. основной сценарий предусматривал использование верификации как средства доказательства корректности процедуры перекачки данных, а не как средства диагностики.

Обзор как доступных для приобретения, так и бесплатных инструментов не выявил удовлетворяющих всем вышеперечисленным требованиям.

4. Реализация

4.1. Контролируемые параметры

Верификация данных выполняется по следующим параметрам:

1. По метаданным — контроль наличия всех необходимых объектов исходной БД в целевой БД и нахождения их в корректном статусе (процедуры успешно откомпилированы, триггеры установлены).
2. Контроль числа строк в таблицах.
3. Хэши столбцов таблиц и таблиц целиком.

Первый из вышеперечисленных пунктов реализуется анализом кодов завершения и других записей в log-файлах, описывающих процесс выгрузки-загрузки. Остальные – сравнением результатов выполнения скриптов подсчета хэшей в исходной и целевой БД.

4.2. Этапы верификации миграции

1. Выполнение скриптов расчета хэшей в исходной БД. Хэши сохраняются в новую таблицу исходной БД.
2. Выгрузка таблицы с хэшами в промежуточный файл.
3. Загрузка результатов из промежуточного файла в целевую БД.
4. Выполнение скриптов расчета хэшей в целевой БД. Результаты пополняют таблицу с хэшами исходной БД.
5. Генерация отчета о расхождениях или полном совпадении набора хэшей.

Шаги, относящиеся к целевой БД, выполняются до включения в журналирующих триггеров.

4.3. Расчет хэшей

Для каждой таблицы вычисляются $n+2$ хэша, где n – количество столбцов. Первый - количество записей в таблице. Несовпадение количества записей позволяет выявить грубые ошибки перегрузки, но совпадение возможно при искажениях в значениях данных, что является большим недостатком, особенно для таблиц, содержащих типы данных, трансформирующиеся при миграции.

Остальные хэши предназначены для устранения этого недостатка. Они рассчитываются следующим образом: записи таблицы обрабатываются одна за одной в произвольном порядке – выполняется запрос $SELECT\,COLUMN_1$, $COLUMN_2$,$\cdots$, $COLUMN_N$ $FROM\ TABLE$.

Значение каждого поля преобразуется в массив байтов, например, datetime преобразуется в количество миллисекунд с 01.01.1970 00:00:00 (POSIX time) как bigint. Получившееся число интерпретируется как массив байтов. Массивы конкатенируются. Получившийся массив подается на вход хэш-функции. Может быть использована любая, реализация которой доступна в обеих СУБД. Авторами была выбрана MD5 [13]. Возвращенное значение представляет собой хэш записи. Данная операция может интерпретироваться как создание дополнительного столбца в таблице. Набор хэшей таблицы не должен зависеть от порядка обхода записей, поэтому было принято решение использовать коммутативную операцию XOR (исключающее или) для получения итоговых хэшей из хэшей записей: для каждого столбца, включая "виртуальный" столбец хэшей записей, вычисляется XOR всех элементов.

MD5 – криптографическая хэш-функция, поэтому возвращаемое значение можно рассматривать как равномерно распределенную случайную величину из интервала $[0; 2^{128} - 1]$. XOR таких величин не меняет распределение, поэтому XOR "виртуального" столбца хэшей записей можно рассматривать как равномерно распределенную случайную величину из интервала $[0; 2^{128} - 1]$. В исходной БД количество записей в каждой таблице не превышает 10^{13}. Вероятность обнаружения не менее одной коллизии не превышает 10^{-12} [14]. Четное количество идентичных записей из-за свойства $XOR(x, x) \equiv 0$ может скрыть ошибку в миграции какого-либо типа данных, но отладка метода производилась с использованием таблиц, содержащих уникальные записи. Вероятность же нескольких аппаратных сбоев, не попавших в журнал, и, повлиявших только на четное количество идентичных записей в одной таблице, оценивается как пренебрежимо малая.

Таблицу БД с первичным ключом можно рассматривать как множество. Изложенный выше метод отличается от *MSet-XOR-Hash* [4] для хэширования (мульти)множеств отсутствием рандомизации – хэш-функция фиксирована, а не выбирается из некоторого семейства. Это делает метод уязвимым для целенаправленной атаки. Однако во время разработки он позволил выявить ряд ошибок в процессе перегрузки.

Остальные хэши вычисляются не криптографическими хэш-функциями, но на этапе отладки они облегчали процесс поиска источников ошибок.

Поскольку данные могут подвергнуться трансформации при перегрузке и представляться разными наборами байтов, вычисление хэш-функции может завершиться с разными результатами в исходной и целевой БД. Для унификации результатов значения полей приводятся к нормальной форме для составления аргумента хэш-функции по правилам, изложенным в

таблице 1.

4.4. Генерация скриптов

Скрипты, не зависящие от схемы БД, – командные файлы для утилит загрузки-выгрузки данных обеих СУБД, SQL-запросы для создания таблиц с результатами и генерации отчета, вспомогательные функции для хэширования отдельных типов данных в исходной СУБД, Java-функция хэширования в целевой СУБД, написаны вручную. Скрипты, зависящие от схемы, генерируются разработанной утилитой (язык реализации F#) на основании актуальной схемы исходной БД (используется экземпляр класса Server) и дополнительных параметров, перечисленных в конфигурационном файле.

Скрипт для исходной БД – код на Transact-SQL определяет три вспомогательных функции: для конвертации DATETIME в BIGINT, вычисления хэша BINARY, вычисления хэша VARCHAR, затем для каждой таблицы вычисляются хэши по схеме, описанной выше.

Скрипт для целевой БД – код на PL/SQL создаёт для каждой таблицы объект CURSOR, передаёт его в Java-функцию вычисления хэшей, сохраняет возвращенный результат в таблицу. Данный подход обусловлен недостаточной производительностью хэширования, реализованного средствами PL/SQL. Как сказано выше, в качестве коммутативной операции, обеспечивающей независимость итоговых хэшей от порядка просмотра записей, выбрана операция XOR, которую в данной версии PL/SQL можно выразить только посредством нескольких вызовов встроенных функций.

Получившийся объем разработанного кода составил 1,5KLOC на F#, 0,3KLOC на cmd/shell, 0,4KLOC на Java. Объем сгенерированного кода для мигрируемой БД, содержащей 2410 таблиц, – 144KLOC на PL/SQL (8,4Mb) и 215KLOC на T-SQL (18,3Mb).

4.5. Показания производительности

Сравнение БД описанным методом требует значительных вычислительных ресурсов. В отличие от загрузки-выгрузки “узким местом” является не производительность системы ввода-вывода, а скорость вычисления хэшей. Затраты времени на сравнение стали сравнимы с временем перегрузки БД, после того как утилита генерации скриптов была доработана для генерации скриптов, вычисляющих наборы хэшей для нескольких таблиц параллельно.

Для MS SQL Server 2005 генерируются несколько файлов с кодом на Transact-SQL для обработки набора таблиц сопоставимого суммарного размера. Каждый скрипт передаётся для исполнения экземпляру служебной утилиты *Microsoft ® SQL Server Command Line Tool* средствами ОС.

Для Oracle 11gR2 генерируется один файл с кодом на PL/SQL, использующий пользовательские функции реализованные на встроенном языке Java для обработки переданного ResultSet. Скрипт передаётся для исполнения служебной утилите *Oracle SQL*Pus*. Средствами СУБД Oracle устанавливается количество потоков, обрабатывающих задания из очереди, заполняется очередь заданий. Каждое задание в очереди обрабатывает одну таблицу или часть таблицы, имеющей численный первичный ключ.

Тип в исходной БД	Тип в целевой БД	Преобразование
BIT	NUMBER(1)	4 байта little-endian
TINYINT	NUMBER(18)	4 байта little-endian
SMALLINT	NUMBER(18)	4 байта little-endian
INT	NUMBER(18)	4 байта little-endian
BIGINT	NUMBER(19)	8 байтов little-endian
FLOAT	FLOAT	8 байтов little-endian
DECIMAL(n, m), $n - m \leq 18$ и $m \leq 18$	NUMBER(n, m)	по 4 байта целой и дробной частей, рассматриваемых как INT, little-endian
DECIMAL(n, m), остальные	NUMBER(n, m)	результат вычисления хэш-функции(MD5) от значения поля, преобразованного в строку
NUMERIC(n, m)	NUMBER(n, m)	аналогично DECIMAL(n, m)
MONEY	NUMBER(19, 4)	аналогично DECIMAL(19, 4)
DATETIME	DATE	аналогично BIGINT после конвертации в Unix time
SMALLDATETIME	DATE	аналогично BIGINT после конвертации в Unix time
BINARY	BLOB	результат вычисления хэш-функции (MD5) от значения поля
IMAGE	BLOB	результат вычисления хэш-функции (MD5) от значения поля
VARBINARY	BLOB	результат вычисления хэш-функции (MD5) от значения поля
CHAR(n), $n \leq 4000$	VARCHAR2(n)	удаляются ведущие и замыкающие пробелы, символы переноса строки (при загрузке в целевую БД происходит нормализация); получившаяся строка конвертируется в набор байтов с учетом кодовой страницы; вычисляется значение хэш-функции(MD5)
CHAR(n), пустая строка или NULL	VARCHAR2(n)	СУБД Oracle рассматривает пустую строку как NULL; набор байтов совпадающий со значением хэш-функции при аргументе – пустой строке
NCHAR(n), $n \leq 4000$	VARCHAR2(n)	аналогично CHAR
SYSNAME	VARCHAR2(128)	аналогично NCHAR(128)
VARCHAR(n), $n \leq 4000$	VARCHAR2(n)	аналогично CHAR
NVARCHAR(n), $n \leq 4000$	VARCHAR2(n)	аналогично CHAR
CHAR(n), $n > 4000$	CLOB	аналогично CHAR
NCHAR(n), $n > 4000$	CLOB	аналогично CHAR
VARCHAR(n), $n > 4000$; VARCHAR(MAX)	CLOB	аналогично CHAR
NVARCHAR(n), $n > 4000$; NVARCHAR(MAX)	CLOB	аналогично CHAR
TEXT; NTEXT	CLOB	аналогично CHAR

Таблица 1: Преобразование полей записей в набор байтов

При выборе подхода к реализации были рассмотрены различные подходы к реализации и проведено сравнение их производительности:

- Код на Transact-SQL в целом удовлетворял требованиям по производительности.
- Реализация хэширования на PL/SQL, – через проход циклом по курсору, с подсчётом хэшей, и их объединение через операцию XOR, реализованную через встроенную функцию BitAND, – проигрывал в производительности TSQL-коду в 5-10 раз. Поэтому было принято решение о его оптимизации.
- Реализация на языке Java, встроенном в Oracle, на разных таблицах показала или сходный результат с PL/SQL-реализацией, или улучшение производительности.
  Дополнительным аргументом, в пользу реализации на Java, так же можно назвать относительною простоту реализации, что позволило быстрее получить как первые, так и окончательные результаты.
- Реализация на языке Java, запущенная в несколько потоков, показала почти линейный рост производительности до достижения насыщения.

На следующем графике представлены сравнение времени работы различных реализаций хэширования для одной из основных таблиц. В уменьшенной для целей тестирования базе в этой таблице 4 миллиона записей общим объемом порядка 2.5 гигабайт. Тестирование производилось на сервере с 8-ядерным процессором.

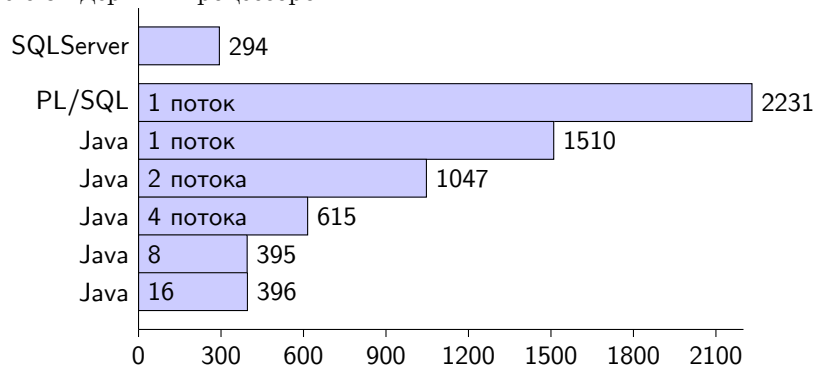

График 1: влияние реализации на время расчета (сек.)

По результатам тестирования для Oracle было принято решение в пользу Java-реализации с запуском несколько потоков.

Итоговые составляющие времени перегрузки тестовой базы, содержащей 21Гб в 1.5К не пустых таблицах даны в таблице 2.

Таким образом, время хэширования сопоставимо со временем перегрузки, что позволяет утверждать, что поставленные требования были удовлетворены.

5. Результаты

Описанным методом сравнивались исходная и целевая БД при каждом ночном тестировании очередной сборки, что позволило выявить и исправить

Этап	Время (сек.)
Выгрузка из MSSQL	1109
Хэширование в MSSQL	799
Загрузка в Oracle	2134
Хэширование в Oracle	2279

Таблица 2: Время этапа миграции тестового набора данных

ошибки в процедуре перегрузки и процедуре вычисления хэшей.

Для доказательства эквивалентности изменений, совершаемых в исходной и целевой БД, при воспроизведении трасс типичных действий пользователя были сгенерированы журналирующие триггеры, вычисляющие хэши перехваченных наборов записей по описанной выше схеме.

6. Заключение

Примененный метод сравнения содержимого баз данных путем подсчёта использующий коммутативные и криптографические хэш-функции позволил успешно решить поставленную перед авторами задачу: было реализовано автоматизированное тестирование корректности миграции данных между различными СУБД путем сравнения набора хэшей, что позволило устранить ряд методических ошибок перегрузки. Разработанный метод возможно переиспользовать, при необходимости пополнив поддержкой типов данных, не встретившихся в схеме исходной БД, и поддержкой СУБД, отличных от MS SQL Server 2005 и Oracle 11gR2.

Список литературы

[1] Maatuk A. *Migrating Relational Databases into Object-Based and XML Databases*. Northumbria University, 2009.

[2] Ulrich Arndt. How to compare tables between td systems in a dual active environment. In *The 2012 Teradata PARTNERS Conference*, 2012.

[3] Mihir Bellare, Oded Goldreich, and Shafi Goldwasser. Incremental cryptography: The case of hashing and signing. In *In CRYPTO*, 1994.

[4] Dwaine Clarke, Srinivas Devadas, Marten van Dijk, Blaise Gassend, and G. Edward Suh. Incremental multiset hash functions and their application to memory integrity checking. In *In Advances in Cryptology - Asiacrypt 2003 Proceedings, volume 2894 of LNCS*, pages 188–207. Springer-Verlag, 2003.

[5] Fabien Coelho. Remote comparison of database tables technical report a/375/cri, 2006.

[6] Fabien Coelho. Remote comparison of database tables. In *DBKDA 2011 : The Third International Conference on Advances in Databases, Knowledge, and Data Applications*, 2011.

[7] Hudicka J. *The Complete Data Migration Methodology.* Dulcian Inc., 2000.

[8] I. Kirilenko and E. Baranov. Automation of QA in the project of DB migration from SQL Server into Oracle. In *Proceedings of the 6th Spring/Summer Young Researchers' Colloquium on Software Engineering (SYRCoSE 2012)*, pages 178–181. Perm, Russia, May 30-31, 2012.

[9] Chang-Yang Lin. Migrating to relational systems: Problems, methods, and strategies. 4(4):369–380, 2008.

[10] R. Guerin M. Bellare and P. Rogaway. Xor macs: New methods for message authentication using finite pseudorandom functions. In *In CRYPTO*, 1994.

[11] Haller K. Matthes F., Schulz C. Testing & quality assurance in data migration projects. In *2011 27th IEEE International Conference*, 2011.

[12] Augustinez J. Redlichx A. Lodha S. Vin H. Deshpande A. Gharote M. Mehrotrak A. Patil S., Royy S. Minimizing testing overheads in database migration lifecycle. In *The 16th International Conference on Management of Data (COMAD)*, 2010.

[13] Ronald L. Rivest. The md5 message-digest algorithm. Internet RFC 1321, April 1992.

[14] Bruce Schneier. *Applied Cryptography, Second Edition.* John Wiley & Sons, 1996.

[15] Bisbal J. Grimson J. Wade V. O'Sullivan D. Richardson R. Wu B., Lawless D. Legacy system migration : A legacy data migration engine. In *Proccedings of the 17th International Database Conference*, pages 129–138. Springer-Verlag, 1997.