УДК 004

*АМОНС О.А.*

# СИНХРОНІЗАЦІЯ ДАНИХ В РОЗПОДІЛЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМАХ

В статті запропоновано механізм синхронізації даних у розподілених системах з урахуванням типів роботи з таблицями та інтеграцією елементів версійності даних. Дослідження присвячено вирішенню завдання підтримки актуальності даних на окремих вузлах розподіленої системи. Проаналізовано особливості роботи розподіленої системи, в рамках якої кожен окремий вузол має локальну копію даних, з якими він працює, та періодично синхронізує данні з центральним сервером. Запропоновано загальний підхід для вирішення проблеми забезпечення актуальності та несуперечливості стану даних кожного користувача системи засобами ведення контролю версій даних та вирішення конфліктів, що виникають в процесі проведення синхронізації. Розглянуто моделі, що використовуються для реалізації підходу.

In the article proposes a data synchronization mechanism in distributed systems based on the types of work with tables and the integration of data versioning elements. Investigation focused on solving tasks support the relevance of the data on the individual nodes of a distributed system. Analyzed the distributed system features in which each individual node has a local copy of the data with which it works and periodically synchronize data with a central server. A general approach has been proposed to solving the problem of ensuring relevance and coherence state of the information for each user of the system by means of doing versioning control of data, recording the changed data and resolving conflicts that arise in the process of synchronization. The models used for the approach implementation have been considered.

## Вступ

В процесі функціонування будь-якої системи центральну роль відіграють дані якими вона оперує. Більшість існуючих програм використовують бази даних в якості сховища для довготривалого зберігання інформації користувача. Більшій частині систем, що орієнтовані для використання на ПК притаманна архітектура де БД винесена на окремий сервер. Однак такий підхід доцільно використовувати лише у випадку наявності стабільного підключення до мережі інтернет.

Мобільні пристрої поширені у всьому сучасному бізнес-середовищі, а це означає обов'язковість роботи з таблицями баз даних, які підключаються лише час від часу. Це спричинено тим, що в процесі їх використання не одноразово виникають ситуації коли з'єднання з мережею інтернет зникає, або взагалі здійснюється лише час від часу. Тому в процесі проектування та розробки системи необхідно передбачити можливість її використання в автономному режимі. З точки зору системи баз даних проблема полягає в ефективності оновлення сховища даних на пристрої, котрі підключаються не часто. [1]

В наш час інформаційна інфраструктура систем приймає все більш розподілений характер. Все більшої децентралізації вимагають рівні управлінських рішень та контроль і керування інформаційними ресурсами [2 – 7]. Структура сучасних організацій являє собою розгалужену систему, до складу якої входять десятки територіально розподілених підрозділів, які функціонують незалежно один від одного, однак використовують одні й ті ж дані. Часто якість функціонування такої складної системи багато в чому залежить від застосування новітніх технологій.

Враховуючи подібні обмеження перед розробниками програмного забезпечення постала проблема організації розповсюдження інформації в системах, які розгорнуто на розподілених клієнтських вузлах. Кожен клієнт повинен мати змогу оперувати актуальними даними. Це означає, що замість зберігання інформації в єдиному сховищі потрібно розповсюдити їх копії по кінцевих вузлах та налагодити механізм синхронізації змін, які будуть вноситись в процесі роботи клієнтських систем.

Питання синхронізації даних на різних вузлах розподіленої система має поширений, однак мало формалізований характер. Тому його огляд та дослідження має важливе значення для пошуку ефективних способів приведення локальних даних в актуальний несуперечливий стан.

Актуальність та доцільність даного дослідження полягає в необхідності створення підсистеми, яку можна було б інтегрувати до складу розподіленої системи для забезпечення син-

хронізації даних між її вузлами. Завдяки використанню такого інструменту з'явиться можливість підтримки даних зв'язаних вузлів розподілених систем в актуальному стані.

### Загальний огляд області дослідження та огляд існуючих рішень

Останнім часом інформаційні технології стрімко розвиваються: з'являються нові пристрої, технології, принципи проектування систем та нові платформи; спостерігається переміщення від стаціонарних комп'ютерів до мобільних пристроїв. Кількість провайдерів та якість підключення до мережі інтернет значно покращились за останні роки, однак далеко не завжди користувачам доцільно тримати підключення активним. Це означає, що до проектування та реалізації розподілених програмних систем ставляться нові вимоги: вони повинні давати можливість користувачам взаємодіяти з актуальними даними і при цьому вміти працювати в режимі, коли підключення до інтернету з'являється лише час від часу. Для мобільних телефонів, планшетів, ноутбуків та інших представників портативних пристроїв завжди було актуальним питання часу їх автономної роботи, причому він залежить не лише від параметрів апаратної складової, але й від програмної частини, що використовується користувачем. Все це створило проблему синхронізації даних між розподіленими клієнтськими системами, яка мінімальним чином використовувала б мережевий трафік та обчислювальні ресурси пристрою.

Проблема синхронізації даних (і саме поняття синхронізації даних) досить широка та не чітка, тому для початку оглянемо визначення, які з нею пов'язані.

Синхронізація даних – процес ліквідації відмінностей між двома копіями даних. Передбачається що раніше ці копії були однаковими, а після цього одна з них, або обидві були змінені незалежно одна від одної. [8]

В [9, 10] синхронізацію визначають як процес генерування та відтворення кількох копій даних, що розміщені на одному чи кількох сайтах, в [2] під синхронізацією даних СУБД розуміють приведення баз даних, що функціонують в розподіленому середовищі, в актуальний стан за рахунок виявлення змінених даних, передачі та збереження цих змін в базі даних отримувача.

Для вирішення цієї задачі часто використовують засоби реплікації баз даних, які вбудовано в обрану СУБД (таких як MS SQL, Oracle та ін.). Це дає потужний засіб для поширення змінених даних на всіх кінцевих точках. Однак головним недоліком такого способу є використання великої кількості обчислювальних та мережевих ресурсів.

Крім того використання вбудованих засобів синхронізації СУБД виключають можливість розробки гетерогенних систем, що є достатньою причиною для пошуку альтернативних способів вирішення подібної проблеми, оскільки спектр пристроїв та платформ на сьогоднішній день надзвичайно широкий.

Застосування технології тиражування даних [4, 6, 7] може виявитись важливим етапом в успішній роботі розподіленої системи. Застосування цієї технології забезпечує гарантовану доставку, своєчасну та цілісну передачу даних. Синхронізація даних передбачає їх дублювання в різних вузлах. При цьому робота системи виконується з локальною базою даних, оскільки вони розміщуються завжди на тому вузлі, на якому обробляються і всі транзакції в системі виконуються локально.

Система синхронізації даних дозволить проектувати та розробляти системи, котрим необхідно підтримувати актуальний стан даних при відсутності стабільного підключення до мережі інтернет. При використанні системи з'явиться інструмент, що дозволить проводити сеанси синхронізації даних оптимальним чином та з використанням не великої кількості обчислювальних ресурсів.

В процесі синхронізації даних неодноразово можуть виникати проблемні ситуації, коли зміни в клієнтському сховищі даних конфліктують зі змінами на центральному сервері. Система синхронізації даних дозволить розв'язувати подібні ситуації та мінімізувати вірогідність їх виникнення.

Задачею системи синхронізації даних є автоматичне виявлення змінених даних та розповсюдження цих змін по всіх вузлах розподіленої системи. Передбачається, що даний спосіб синхронізації буде доречним для використання на мобільних пристроях, тому для запланованої системи потрібно обрати максимально швидкі та енергоекономічні методи синхронізації і пошуку змінених даних в базі даних клієнта.

Синхронізацію можна класифікувати по різному. Для проектування та подальшої

розробки системи синхронізації варто чітко визначитись з основними властивостями та шляхами її проведення. В рамках даного дослідження було виділено наступну структуру класифікації [7, 9 – 15]:

1. За напрямком:

Однонаправлена, або одностороння синхронізація – дані змінюються тільки в одній з БД, а в іншій лише зберігаються і не підлягають змінам.

Мультинаправлена, або багатостороння синхронізація – дані можуть змінюватись та вводитись у всіх БД.

2. За часом проведення синхронізації:

Синхронізація реального часу, або синхронна синхронізація – тип синхронізації за якої дані повинні бути синхронізовані негайно після змін.

Відкладена, або асинхронна синхронізація – тип синхронізації при якій процес синхронізації запускається після певної події. Тобто зміни в одній з копій даних поширюється на інші після певного проміжку часу, а не в тій же транзакції.

3. За способом передачі інформації під час синхронізації:

Пряма синхронізація – синхронізація виконується за допомогою програми-клієнта, котра має стабільне підключення до сервера БД.

Непряма, або недетермінована синхронізація – синхронізація виконується за допомогою програми-клієнта, що не має стабільного підключення до сервера БД.

4. За способом аналізу інформації, що синхронізується:

Синхронізація за станом – алгоритм синхронізації працює за принципом порівняння записів однієї таблиці з записами іншої і на основі цієї інформації приймається рішення про проведення синхронізації.

Синхронізація змін, або дельта синхронізація – алгоритм синхронізації виконує перенесення лише змін інформації БД, які зберігаються в журналі виконаних транзакцій.

**Ведення версій та історії даних.** При розробці баз даних зазвичай необхідно забезпечити підтримку обліку версій та збереження історії об’єктів. Наприклад, у працівника підприємства може змінитися посада, у посади в свою чергу може змінитись оклад – в багатовимірному моделюванні це називається Slowly changing dimensions (далі SCD) – виміри, що рідко змінюються. Тобто виміри, не ключові атрибути яких мають тенденцію з часом змінюватись. Всього виділяють 6 основних типів (методів) SCD, що відображають як історія змін може бути відображена в моделі [16]. Кожен з методів передбачає введення додаткових полів та таблиць до записів проблемної області для відслідковування їх історії зміни. Проведемо короткий огляд кожного типу.

*Тип 0* полягає в тому, що дані після першого збереження в таблицю більше ніколи не змінюються. Цей метод практично майже ніким не використовується оскільки він не підтримує ведення версій. Він потрібен лише як нульова точка відліку для методології SCD.

*Тип 1* – це звичайний перезапис старих даних новими. В чистому вигляді цей метод теж не містить ведення обліку версій і використовується лише там де історія фактично не потрібна. Тим не менше в деяких СУБД для цього типу можна додати обмежену підтримку версійності засобами самої СУБД (наприклад Flashback query в Oracle), або відслідковуванням змін через тригери.

*Тип 2* полягає в створенні для кожної версії окремий запис в таблиці з додаванням полякючового атрибута даної версії, наприклад: номер версії, дата зміни чи дата початку та закінчення періоду існування версії.

*Тип 3* полягає у тому, що в самому записі знаходяться додаткові поля для минулих значень атрибута. При отриманні нових даних старі дані перезаписуються поточними значеннями.

*Тип 4* передбачає збереження історії змін в окремій таблиці: основна таблиця завжди перезаписується поточними даними з перенесенням старих даних в іншу таблицю. Зазвичай цей тип використовують для обліку змін або створення архівних таблиць. Підтипом або гібридом цього варіанта (з типом 2) слід вважати секціонування по признаку поточної версії з дозволеним переміщенням рядків, але це вже за границею моделювання та скоріше відноситься до адміністрування.

*Тип 6 (1+2+3)* – так званий гібридний тип був придуманий Ральфом Кімболлом (Ralph Kimball) як комбінація методів, що описано вище, та призначений для ситуацій, які вони не враховують або для більшої зручності роботи з даними. Він полягає у внесенні додаткової збитковості: береться за основу тип 2, додається сурогатний атрибут для альтернативного обзору версій (тип 3) та перезаписуються одна, або всі попередні версії (тип 1). В цілому будь-яка

комбінація основних типів SCD відноситься до гібридного типу, тому як їх недоліки так і переваги залежать від конкретної реалізації.

**Методи обліку змінених даних.** Для розробників однією з проблем являється відслідковування того, які дані було змінено в базі. Ще більш серйозною задачею являється створення простого рішення, яке не наносило б серйозного удару по продуктивності робочих навантажень та було б не складним в створенні, реалізації та керуванні.

Мобільні пристрої поширені у всьому сучасному бізнес-середовищі, а це означає обов'язковість роботи з таблицями баз даних, які підключаються лише час від часу. З точки зору системи баз даних проблема полягає в ефективності оновлення сховища даних на пристрої, котрі не мають стабільного підключення до інтернету. Наявність локальної копії даних вирішує дану проблему та надає користувачу можливість використовувати систему в режимі оф лайн (тобто без наявності активного підключення до інтернету) так, ніби він взаємодіє з центральним сховищем даних. Однак в цьому випадку виникає нова проблема – оновлення локального сховища даних із внесенням змін, що були виконані на центральному сервері з моменту останньої синхронізації, вирішенням конфліктів, що виникли у зв'язку з невідповідністю локальних і серверних даних, та з передачею власних змін серверу для поширення на інших вузлах системи. Для всіх перерахованих пунктів ключовою являється задача пошуку замінених даних як в локальній базі даних, так і на серверній стороні. Передача всіх записів бази даних від одного вузла до іншого в процесі синхронізації – не найкраще рішення. Воно тягне за собою фінансові затрати та затрати обчислювальних ресурсів, причому це стосується всіх учасників обміну інформацією: клієнтського пристрою, каналу зв'язку та серверної частини (або ж пристрою іншого клієнта, в залежності від обраної архітектури системи та принципів проведення синхронізації). Наприклад, мобільні пристрої мають обмежений час автономної роботи (мається на увазі роботу без під'єднаного зарядного пристрою), тому це вимагає від розробників максимально ефективно оптимізувати свої програмні продукти, оскільки будь-яке виконання зайвих обчислень пришвидшує зменшення заряду акумулятора пристрою. Тобто передаючи весь обсяг інформації бази даних (так званий моментальний знімок даних – snapshot) відбувається неефективне використання ресурсів. По каналу зв'язку будуть передаватись великі обсяги даних, навіть якщо в них буде змінено лише одне поле, а отже мережевий трафік та плата за нього значно зростуть. А у разі використання мобільного, чи не стабільного інтернет з'єднання з'являється висока імовірність того, що сеанс синхронізації завершиться невдачею. Щодо серверної частини то тут теж присутні обмеження. Використовуючи хмарні обчислення в своїх рішеннях необхідно орендувати в провайдера головним чином процесорний час, обсяг сховища бази даних та оплачувати виконання транзакцій до нього. Кожне рішення, що не оптимальним чином використовує вказані ресурси тягне за собою неоправдані фінансові витрати. Для вирішення повідних проблем більш доцільним є передача лише тих записів, що зазнали змін на вузлі-джерелі з часу останнього сеансу синхронізації. Враховуючи це необхідно включити до системи синхронізації даних механізм пошуку змін, який буде виділяти лише змінені дані з-поміж всіх інших записів бази даних.

Розглянемо існуючі методи для виявлення змінених даних. В рамках даної роботи було умовно їх розділено на журнальні методи та методи пошуку змін за запитом. Перша група методів передбачає збереження дій, що виконуються з записами бази даних до журналу та їх відтворення на інших вузлах синхронізації. При використанні другої групи – пошук змінених даних відбувається безпосередньо під час проведення сеансу синхронізації, або перед ним.

У випадку застосування методу ведення журналу змінених даних система реєструє в журналі інформацію про зміну даних в таблицях користувача, відслідковуючи як сам факт виконання операцій по зміні даних так і конкретні зміни. Як показано на рис. 1 зміни в таблиці користувача записуються у відповідних таблицях змін. Ці таблиці забезпечують журнальне представлення виконаних змін, що розподілені в часі.

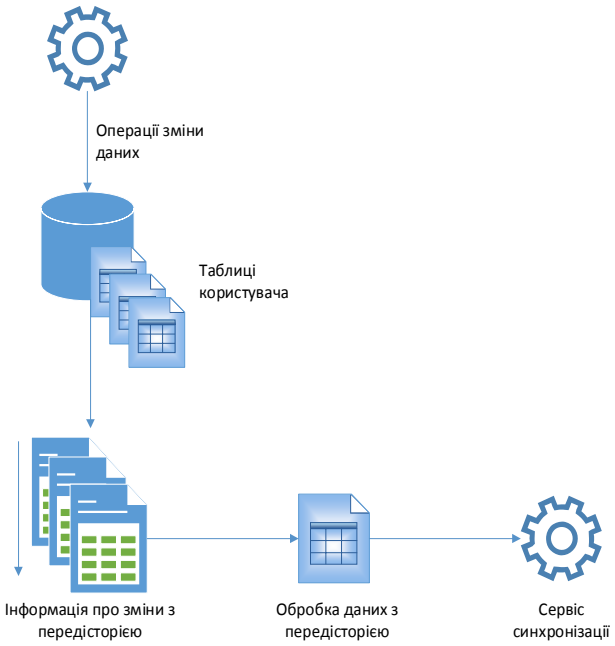


***Рис. 1. Використання журналу змінених даних***


***Рис. 2 Використання журналу змін***

Також можливий варіант, коли в журнал вносяться не змінені дані, а сам факт про зміну рядків таблиці (рис. 2). Це дозволяє програмним системам визначати змінені рядки, отримуючи інформацію про зміни безпосередньо з таблиць користувача. Тобто в журналі міститься інформація про те, які рядки таблиць змінені без додаткової інформації про те, які поля було змінено. Процес вибірки змінених рядків відбувається безпосередньо перед проведенням процесу синхронізації. При цьому відбувається проходження по записах журналу та вибір відповідних рядків з таблиць бази даних.

В цьому сценарії на сервіс синхронізації передаються всі рядки таблиці, що змінено з моменту проведення останньої синхронізації, причому передаються поточні дані рядків цілком. Оскільки, як вже було сказано, для проведення синхронізації використовується синхронний механізм для відслідковування змін, система може виконати двосторонню синхронізацію і визначити можливі конфлікти. Однак слід зауважити, що використовуючи даний підхід слід мати на увазі, що у разі виникнення конфлікту буде вкрай складно реалізувати його розв'язання шляхом об'єднання даних.

В [17] розглянуто синхронізацію журналу транзакцій – найбільш простий в реалізації метод та один з найефективніших і найпоширеніших. В цьому випадку до журналу записуються транзакції бази даних, що були виконані впродовж певного часу. При реалізації процесу синхронізації двох та більше БД таким способом на кожній з них повторюються всі дії, що виконані на іншій БД, витягуючи їх по черзі з журналу. Звідси випливає одна з основних переваг синхронізації журналу транзакцій – немає необхідності в постійному з'єднанні з віддаленою базою даних під час передачі змін – частина журналу являється автономною одиницею і її можна передавати будь-якими каналами зв'язку в будь-який час, в тому числі навіть на зовнішньому флеш-накопичувачі. Тобто журнал володіє властивістю переносимості, яку доволі важко реалізувати у разі використання інших методів.

Для реалізації цього метода необхідно ретельно враховувати які записи журналу ще не передавались. Ця проблема стає складнішою при наявності кількох баз даних, що синхронізуються – потрібно запам'ятовувати який останній запис було передано в кожну з БД.

Недоліком використання журналу транзакцій являється те, що журнал займає доволі багато місця. При достатньо частій зміні інформації в таблицях бази даних і достатньо великих проміжках часу між сеансами синхронізації журнал може сягати великих розмірів і навіть перевищувати за розміром самі дані, що зберігаються в базі даних. Зрозуміло, що після успішного застосування журналу до іншої БД журнал можна урізати. Однак тим не менше журнал за-

ймає доволі багато місця – в кращому випадку дублює нові та змінені записи.

Журнальна синхронізація реалізується відносно просто, однак колізії (конфлікти) при цьому неминучі, оскільки процес, що проводить синхронізацію не має одночасного доступу до кількох баз даних, які приймають участь в процесі.

На противагу журнальним методам пошуку змінених даних можна виконувати пошук шляхом порівняння даних двох таблиць, що беруть участь у синхронізації. Одним із методів порівняння блоків даних є визначення значення хеш-функції. Хеш-функція – функція, що перетворює вхідні дані будь-якого (як правило, великого) розміру в дані фіксованого розміру. Відповідно хешування – процес перетворення вхідного масиву даних довільної довжини у вихідний бітовий рядок фіксованої довжини. При цьому, якщо хеш-функції повертають різні значення для двох вхідних масивів, то вони гарантовано різні. Якщо ж хеш-значення двох вхідних масивів співпадають, то вхідні рядки скоріше за все однакові. [18]

У разі застосування цього методу для проведення синхронізації пошук змін в базі даних відбувається для кожної таблиці окремо. Найменшим елементом вважається запис в таблиці. Спочатку обчислюється хеш-сума для всієї таблиці в цілому і якщо це значення на обох вузлах синхронізації виявиться різним, то далі виконується розбиття таблиці на інтервали, для кожного з яких також проводиться визначення та порівняння результатів хеш-функції. Для інтервалів з різними результатами проводиться подальше подрібнення і для кожного проміжку процес повторюється до тих пір, доки не будуть виявлені змінені рядки таблиці.

Для застосування пошуку змін за допомогою хешування не потрібно вносити зміни до існуючої структури бази даних чи додавати логіку до процесу збереження даних в системі. Також не потрібно проводити ведення журналу змін. Однак разом з цим значно підвищуються вимоги до обчислювальних ресурсів на всіх вузлах синхронізації, а у разі проведення синхронізації даних, де змінено невелику кількість записів все одно буде необхідно досліджувати всі записи таблиці, що вкрай неефективно (особливо у випадку великої кількості рядків таблиці).

### Постановка проблем дослідження

В реаліях сучасних інформаційних систем є багато задач на реалізацію рішень взаємодії окремих вузлів розподіленої системи. Окремої уваги заслуговують ті з них, що орієнтовані на мобільні пристрої та роботу за умов з нестабільним підключенням до мережі інтернет. В такому випадку раціональним підходом вважається використання локальної бази даних на пристрої користувача та використання копії даних розподіленої системи (чи копії частини даних), яка зберігається в цій локальній БД. Для реалізації подібного сценарію необхідно спроектувати та розробити механізм виявлення та розповсюдження даних, що змінені на одному з вузлів системі між всіма іншими вузлами. Для цього потрібно знайти вирішення ряду проблем, що виникають впродовж проведення синхронізації даних. Однією з таких проблем є вибір моделі розподіленої системи, характер взаємодії клієнтів та характеристики процесу синхронізації. Враховуючи незалежність роботи локальних СУБД на різних вузлах необхідно забезпечити унікальність первинних ключів записів таблиць. Вкрай недопустимо щоб в різних БД були створені рядки з однаковими ідентифікаторами. Це може призвести до некоректного проведення процесу синхронізації та непередбачуваних наслідків. Також в рамках дослідження необхідно вирішити специфічні проблеми, що безпосередньо стосуються процесу синхронізації даних та притаманні обраній стратегії проведення синхронізації. Ще одним важливим аспектом проблеми є питання пошуку даних, які змінено на окремо взятому вузлі системи з часу проведення останнього сеансу синхронізації, та забезпечення безпеки доступу до даних в БД. І насамкінець лишається найбільш обширна та складна проблема – розробка алгоритму проведення процесу синхронізації. Це питання в більшій мірі визначає наскільки ефективно будуть використовуватись рішення, які було прийнято раніше.

### Запропоноване рішення

Враховуючи вимоги до системи синхронізації та відповідно до наведеної класифікації типів проведення синхронізації вирішено, що процес синхронізації в рамках роботи підсистеми матиме наступні характеристики:

– за напрямком – мультинаправлена: кожна з віддалених БД виступатиме в ролі як джерела так і цілі синхронізації;

– за часом проведення – асинхронна: початком для проведення сеансу синхронізації слугуватиме запит клієнтської системи;

– за способом передачі інформації – недетермінована: проводитиметься час від часу за запитом клієнтської системи.

– за способом аналізу інформації – синхронізація змін: на кожному вузлі проводитиметься ведення версійності та реєстрації змінених даних.

В рамках системи синхронізації даних реалізовано двонаправлену синхронізацію між центральним сервером та кожним з клієнтів. Проводиться журналізація всіх змінених даних. Фізичне видалення та зміна записів замінюється логічним – тому конфлікти стають розв'язуваними.

**Вирішення проблеми забезпечення унікальності ідентифікаторів.** Необхідно зазначити, що кожен вузол системи, що бере учать в процесі синхронізації сам по собі являється системою бази даних. Інакше кажучи, на кожному вузлі є власна локальна СУБД та програмне забезпечення, що з нею пов'язано.

Протягом проведення процесу синхронізації важливо щоб кожен запис мав унікальний ідентифікатор. Тому з метою забезпечення унікальності первинних ключів записів таблиць було вирішено відмовитись від ідеї автоматичного їх генерування засобами СУБД. Дане рішення викликане імовірністю того, що в двох базах даних можуть з'явитись запаси з однаковими ключами (особливо у разі використання автоінкрементованих ключів). Для виходу з ситуації можна використовувати зіставні ключі, однак це додасть збитковості записів та ускладнить операції з даними в системі.

Враховуючи все перераховане вище було вирішено використовувати тип GUID в я кості типу первинних ключів. Генерація нових ключів має відбуватись в самій системі. Це унеможливить імовірність створення двох записів однієї таблиці з однаковими ідентифікаторами та спростить механізм проведення синхронізації та розв'язання конфліктних ситуацій.

**Вирішення проблем двосторонньої синхронізації.** Якщо реалізовувати схему з двосторонньою синхронізацією, тобто коли вузли являються і джерелом і приймачем інформації, що синхронізується, виникає ряд специфічних проблем. Розглянемо випадок лише для двох учасників (рис. 3). При цьому будемо будем мати на увазі, що один з них являється клієнтом (окремо-взятим вузлом системи), а інший – центральним сервером синхронізації, з яким зв'язуються всі клієнти системи.

При подібній схемі реалізації синхронізації даних кожен з вузлів-учасників процесу виступає в якості джерела та приймача даних. В приведеній схемі можуть виникнути проблеми, що описано далі.

По-перше, кожен з вузлів отримує журнал дій, що виконано на іншій стороні.

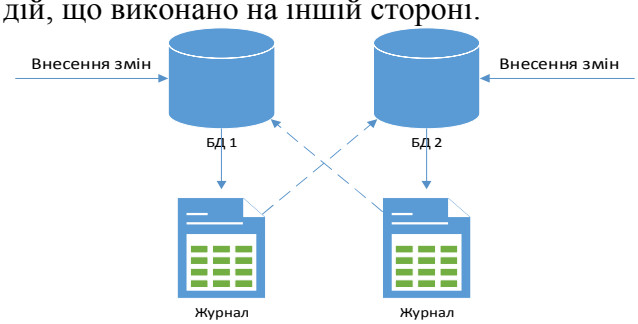


***Рис. 3. Схема двосторонньої синхронізації***

Кожен вузол приймає запити на модифікацію даних як від локальних користувачів, які працюють з базою даних, так і від віддаленої сторони синхронізації. При відтворенні цього журналу вузлом-приймачем виконуються всі записані дії, які було виконано на вузлі-джерелі. Якщо цей вузол-приймач виступає в ролі також і вузла-джерела, то дані, що виконано в процесі відтворення журналу, також потраплять в журнал вузла-приймача і, відповідно, при наступному сеансі зв'язку їх буде передано відправнику знову. Тобто може виникнути процес неконтрольованого лавиноподібного процесу накопичення інформації, яка синхронізується, коли вузли передають один одному інформацію про один і той же запис.

Для запобігання подібної поведінки можна зробити наступне: при прийомі та виконанні журналу, що отримано від іншої сторони, – не вести власний журнал щоб виконані зміни були виконані лише над локальними даними і не потрапили в журнал, що призначено для відправки вузлу, який щойно прислав свій журнал. Якщо для реалізації журнальної синхронізації використовується механізм тригерів, то потрібно блокувати їх виконання під час виконання журналу іншої сторони, або написати таким чином, щоб вони розпізнавали момент виконання журналу іншої сторони і не писали б в такому випадку власний журнал.

Якщо немає блокування ведення журналу інформації, отриманої від партнера синхронізації, виникає ситуація, яку представлено на рис. 4, тобто інформація, що синхронізується ходить від одного вузла до іншого, що звичайно ж недопустимо.

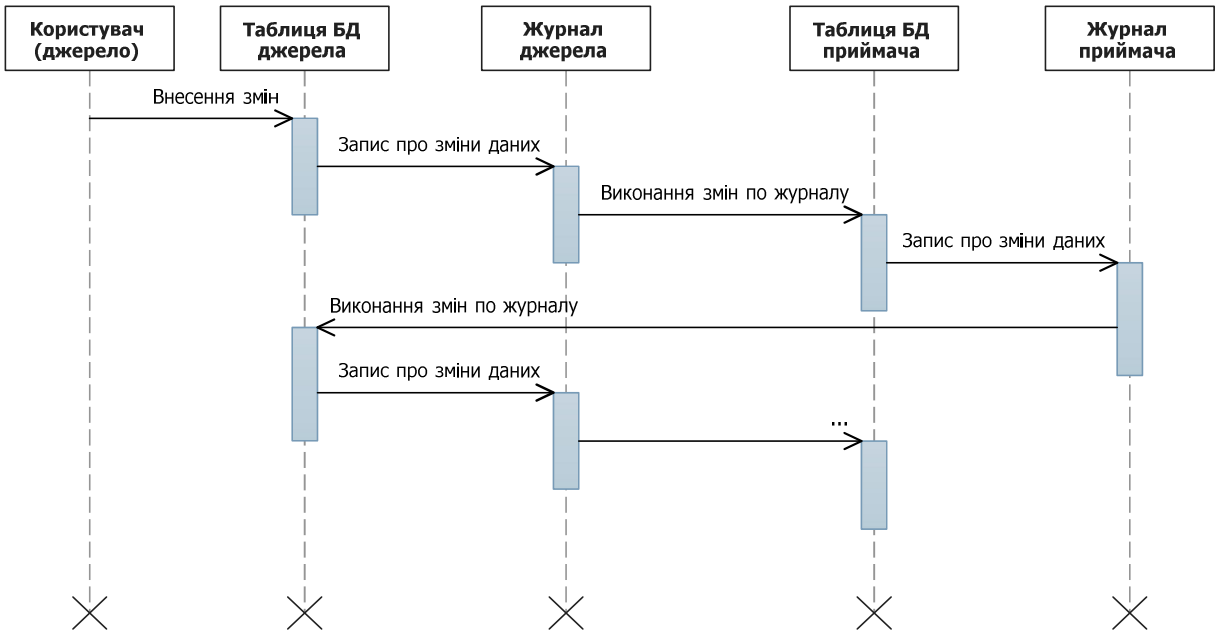


***Рис. 4. Проблема блокування під час проведення двосторонньої синхронізації***

По-друге, при реалізації двосторонньої синхронізації неодмінно виникає проблема конфлікту доступу до одних і тих же даних на обох сторонах учасників процесу синхронізації.

Цей конфлікт виникає у тому випадку, якщо на двох вузлах виконуються зміни одних і тих же даних. Припустимо, на вузлі 1 створили запис, а потім на вузлі 2 її видалили. Однак одночасно з цим видаленням на вузлі 1 було виконано операцію по зміні цього запису. Звичайно ж виникає конфлікт, який потребує вирішення – які зміни рахувати вірними. Основне питання, на яке потрібно дати відповідь при виникненні подібного роду конфліктів, – які дані все ж вірні і хто з вузлів-учасників правий. Відмітимо, що далеко не завжди можливо прийняти автоматизоване рішення про перевагу змін для якогось запису і необхідне рішення користувача.

Для розв'язання цього конфлікту використаємо наступні принципи:

– більш вірними вважаються записи у того з вузлів, який створив запис і хто вважається його власником. Для реалізації цього правила необхідно щоб в кожній з таблиць процесу синхронізації було поле, що вказує на автора запису;

– вводиться додаткове поле до журналу – "час виконання зміни". При виконанні журналу час зміни запису порівнюється з часом локальної зміни цього запису і більш вірною вважається інформація, яка отримана пізніше. Слід зазначити, що це далеко не найкращий спосіб вирішення конфліктів, оскільки може створювати конфлікт зміни одного і того ж запису, що відбувся в різний час;

– одним з найнадійніших способів – це механізм відслідковування за виникненням конфліктних ситуацій, а у випадку виникнення таких ситуацій – вимагати рішення користувача, не вирішуючи яка із сторін права, однак даючи оцінку кожній з таких ситуацій.

**Вирішення проблеми контролю доступу до даних та мінімізації обсягу трафіку.** В рамках роботи системи було вирішено виділити чотири типи сутностей даних, в залежності від яких обмежується доступ до них. Обмеження залежать від користувача, що намагається отримати доступ до даних:

– службові дані: доступ для зміни та перегляду інформації належить лише адміністратору;

– особисті дані: створювати записи такого типу можуть всі користувачі, однак переглядати та редагувати – лише автори записів;

– публічні дані: право на створення та перегляд інформації належить всім користувачам системи, однак редагувати записи може лише автор;

– загальнодоступні дані: створення, перегляд та редагування даних можуть здійснювати всі користувачі системи.

Ці всі обмеження наочно зображено в табл.1.

***Табл. 1 Можливість виконання дій над даними системи в залежності від типу записів таблиці та ролі користувача***

| **Тип користувача** | **Дії** | **Тип запису** | | | |
|---|---|---|---|---|---|
| | | **Службові дані** | **Особисті дані** | **Публічні дані** | **Загальнодоступні дані** |
| Адміністратор | Створення | ✓ | | | |
| | Перегляд | ✓ | | | |
| | Редагування | ✓ | | | |
| | Видалення | ✓ | | | |
| Автор запису | Створення | | ✓ | ✓ | ✓ |
| | Перегляд | | ✓ | ✓ | ✓ |
| | Редагування | | ✓ | ✓ | ✓ |
| | Видалення | | ✓ | ✓ | ✓ |
| Інші користувачі | Створення | × | | | |
| | Перегляд | ✓ | × | ✓ | ✓ |
| | Редагування | × | × | × | ✓ |
| | Видалення | × | × | × | ✓ |

Введемо певні пояснення щодо таблиці: роль адміністратора може бути суміщена з роллю автора запису, або роллю іншого користувача у відношенні до даних, які не являються службовими.

В контексті бази даних типи об’єктів та обмеження поширюються на таблиці БД. Це означає, що, наприклад, якщо в певній таблиці зберігаються об’єкти типу службових даних то доступ до неї матимуть лише користувачі з правами адміністраторів.

Право на редагування записів таблиць зі службовими даними мають лише адміністратори. При цьому не має значення чи являється адміністратор автором запису (користувачів з правами адміністратора може бути декілька і не має значення хто з них додав запис). Всі інші користувачі системи мають права лише на перегляд даних такого типу. Прикладом можуть бути таблиці-довідники.

Доступ до особистих даних: право на перегляд та редагування можуть мати лише автори цих записів. Для всіх інших користувачів дані такого типу повністю скриті.

Суть публічних даних така ж як і в особистих даних, однак користувачі, що не являються авторами можуть переглядати подібні записи.

Загальнодоступні дані повністю відкриті для перегляду та редагування всіма користувачами. Очевидно, що при цьому авторство не має ніякого значення.

Розглянемо більш детально ролі користувачів та типи даних у контексті синхронізації баз даних. Оскільки вже визначено основні характеристики системи синхронізації і вказано, що використовується сервер в якості центрального вузла для проведення синхронізації то всі подальші описи виконано з урахуванням саме прийнятої архітектури.

**Синхронізація службових даних.** Службові дані доступні для перегляду всім користувачам системи, а для редагування – лише користувачам з правами адміністратора. Тому в процесі синхронізації службові дані доступні для читання для всіх вузлів системи, тобто розповсюджуються для всіх користувачів. У разі редагування такого запису клієнтською системою при спробі зберегти зміни на сервері здійснюється перевірка прав користувача: якщо користувач являється адміністратором, то зміни приймаються і в майбутньому будуть розповсюджені по всім іншим вузлам. У тому разі, якщо користувач не має прав адміністратора – зміни не приймаються на сервері, а на клієнті повертаються до попереднього стану.

Важливо відмітити сценарій, коли запис службових даних редагується двома адміністраторами одночасно. В такому разі система синхронізації намагається поєднати зміни. Якщо у обох адміністраторів змінені поля не пересікаються – відбувається поєднання і обидві зміни стають актуальними. Якщо ж змінені поля пе-

ресікаються то актуальними залишаються зміни, що виконані раніше, а корекції другого користувача відкидаються і оновлюються на клієнті.

**Синхронізація особистих даних.** Особисті дані доступні лише автору запису, тому в даному випадку ніяких особливих проблем не виникає. В процесі проведення синхронізації таблиць з особистими даними беруть участь лише ті дані, автор яких проводить синхронізацію. Тобто записи з таких таблиць копіюються лише їх авторам і не відображаються в таблицях інших користувачів. У випадку спроби зберегти на сервері перевіряється авторство кожного запису таблиці. Якщо користувач намагається зберегти запис особистих даних, автором якого являється інший користувач – збереження не відбувається. Якщо ж запис зберігає сам автор то відбувається збереження змінених даних.

**Синхронізація публічних даних.** Ситуація з публічними даними дуже схожа з особистими даними. Відмінність лиш полягає у тому, що публічні дані доступні всім користувачам, тобто вони копіюються на всі вузли системи. У разі редагування такого запису клієнтською системою при спробі зберегти зміни на сервері здійснюється перевірка авторства запису: якщо користувач являється автором, то зміни приймаються і в майбутньому будуть розповсюджені по всім іншим вузлам. У тому разі, якщо користувач не являється автором – зміни не приймаються на сервері, а на клієнті повертаються до попереднього стану.

**Синхронізація загальнодоступних даних.** Найцікавіший і водночас найскладніший випадок складають загальнодоступні дані – вони доступні для читання та редагування всім користувачам системи. Тобто в процесі синхронізації вони поширюються по всіх вузлах системи. У разі редагування такого запису клієнтською системою при спробі зберегти зміни на сервері необхідно передбачити логіку обробки різних ситуацій. У разі якщо клієнт відредагував загальнодоступний запис, який був перед цим актуальним то його зміни приймаються та в майбутньому поширюються по всіх інших вузлах.

**Основні концепції алгоритму проведення синхронізації.** Після аналізу вимог до системи та проблем, що виникають в процесі синхронізації було обрано алгоритм для проведення сеансів синхронізації. Перед його описом слід зазначити кілька важливих моментів.

Ідентифікатори версій генеруються на клієнті, а зберігаються в таблиці на сервері. Це дозволить вести облік версій та відслідковувати хронологію внесення змін. У кожного запису таблиць БД на сервері додано поле в яке слід зберігати відповідний номер.

Для зберігання інформації про змінені поля при кожному збереженні даних на сервері вирішено використовувати *.xml-файли. Назва кожного файлу буде співпадати з ідентифікатором версії даних, в рамках якої збережено зміни. Таким чином можна буде легко знайти необхідні зміни для проведення процесів розв’язання конфліктів та формування пакету змінених даних для відправки клієнту. Використання файлів дозволить зменшити об’єм пам’яті на збереження подібної інформації та дозволить швидше знаходити необхідну інформацію, оскільки одразу відомо абсолютний шлях до файлу в файловій системі серверу.

Розглянемо основні принципи алгоритму проведення синхронізації даних клієнта по кроках:

1. В певний момент часу роботи системи клієнта до підсистеми синхронізації надходить запит на початок проведення сеансу синхронізації даних. Якщо пристрій користувача в цей момент має підключений доступ до мережі інтернет та є можливість зв’язатись з центральним сервером синхронізації то продовжуємо цей процес, якщо ні – повертається помилка про неможливість проведення синхронізації в даних момент.
2. Генеруємо новий ідентифікатор версії стану локальних даних клієнта.
3. Далі на клієнті відбувається формування пакету змінених даних. Якщо з моменту проведення останнього сеансу синхронізації жодні зміни не були внесені – список змінених рядків буде порожнім.
4. Після того як список змінених даних сформований він відправляється на центральний сервер разом з ідентифікатором версії даних останньої синхронізації та ідентифікатором поточної версії, який щойно згенеровано. Якщо синхронізація ще жодного разу не проводилась то версія останньої синхронізації буде порожньою.
5. Отримавши пакет сервіс синхронізації перевіряє чи є в ньому змінені рядки: якщо немає то клієнт не вносив змін в локальну БД, тому переходимо до кроку 10, якщо є – виконуємо процес далі.

6. Якщо версія останньої синхронізації виявилась актуальною (останньою версією на сервері) – отже клієнт працював з поточним станом даних серверу і його зміни точно не конфліктують зі змінами на сервері (Розв'язання конфліктів не потребується тому переходимо до кроку 8).

7. У тому разі якщо на попередньому кроці виявилось, що з моменту коли клієнт синхронізувавсь востаннє були внесені зміни до центрального сховища то перед збереженням його змін потрібно виконати дослідження на наявність конфліктів та розв'язати їх.

Вирішено виконувати розв'язання конфліктних ситуацій методом об'єднання даних (merge). Проводити його будемо наступним чином: з таблиці ідентифікаторів версій вибираємо всі записи, що було здійснено з часу останньої синхронізації клієнта. За цими ідентифікаторами зчитуємо *.xml-файли в яких зберігається інформація про змінені поля рядків. Якщо зміни пересікаються – на основі прийнятих правил вирішуємо якій зі змін дати перевагу.

8. Зберігаємо ідентифікатор нової версії, який було згенеровано раніше користувачем (тепер вона стає актуальною) та ідентифікатор користувача, що вносить зміни, до таблиці версій.

9. Коли конфлікти в даних розв'язано можна переходити безпосередньо до збереження змін в базу даних. Виконувати збереження слід з урахуванням ролі користувача (адміністратор, автор запису, не автор запису) та типу даних, які зберігаються (службові дані, особисті дані, публічні дані, загальнодоступні дані). До кожного збереженого запису у відповідну колонку додається ідентифікатор поточної версії. Паралельно зі збереженням записів в базу даних необхідно створити *.xml-файл, назва якого співпадає з ідентифікатором актуальної версії, та записувати в нього інформацію про змінені поля записів.

10. Після збереження змін користувача необхідно сформувати пакет змінених даних, що відбулися з часу останньої синхронізації клієнта. Перевіряємо версію даних коли востаннє синхронізувався клієнт. Якщо вона порожня, (або ж такої версії не знайдено в таблиці ідентифікаторів версій) то з цим клієнтом ще жодного разу не проводилась синхронізація і вибираємо всі записи з БД, які доступні даному клієнту. В інакшому випадку використовуємо ті ж *.xml-файли, що були вибрані на кроці 7 та формуємо пакет змін для користувача з урахуванням його ролі та типу даних таблиць.

11. Сформований пакет повертаємо на пристрій користувача, а зміни, що містяться в пакеті зберігаємо до його локальної бази даних.

12. Оскільки синхронізація відбулась успішно можна очистити журнал змін на пристрої користувача (вони вже успішно збережені на центральному сервері).

13. Зберігаємо ідентифікатор поточної версії даних. Тепер він буде використовуватись у наступному запиті на синхронізацію.

14. Сеанс синхронізації завершено. Локальні дані користувача в актуальному стані.

Виконуючи синхронізацію за описаним принципом можна досягти розповсюдження інформації та змін в ній між всіма вузлами розподіленої системи. Як видно з алгоритму центральний сервер синхронізації приймає роль пасивного вузла, оскільки кожен з клієнтів посилає запит на проведення процесу оновлення даних на центральний сервіс.

Для реалізації виконання сервером активної ролі можна модифікувати описане рішення, додавши розсилання Push-повідомлень, котрі будуть розсилатись всім розподіленим користувачам та інформувати їх про оновлення даних сервера. Після отримання такого повідомлення система користувача може провести оновлення власних даних. Тобто час між проведенням окремих сеансів синхронізації значно зменшиться, що підвищить достовірність стану локальних даних.

Для більш оптимального використання файлового сховища можна передбачити механізм архівації старих *.xml-файлів. Вірогідність їх використання з часом зменшується, а при необхідності файл легко може бути отриманий з архіву.

Насамкінець потрібно дати важливе зауваження щодо забезпечення надійності збереження даних центральної бази даних та файлів зі зміненими полями: враховуючи важливість цих даних потрібно передбачити їх дублювання та зберігання в декількох місцях. Сучасні хмарні платформи забезпечують достатній рівень надійності зберігання даних, тому вірним рішенням буде використання таких платформ для розгортання серверної частини системи.

## Висновки

В результаті проведеного дослідження запропоновано новий підхід щодо синхронізації даних розподіленої системи на різних її вузлах. Такий підхід перш за все дозволяє реалізовувати механізми забезпечення актуальності даних на різних вузлах системи та поширювати зміни між ними. Основною особливістю даного методу являється розбиття даних системи на типи та здійснення контролю доступу до них в залежності від ролі користувачів. Таке рішення водночас забезпечує безпеку доступу та зменшує обсяг трафіку даних, що передаються по мережі в процесі синхронізації.

Під час дослідження цільової області було проведено детальний аналіз проблем, які притаманні синхронізації та виконано огляд існуючих методів їх розв'язання. Модифікації цих методів, що запропоновані в дослідженні, надають більш ефективний спосіб боротьби з проблемами, а запропонований метод синхронізації в залежності від типу даних взагалі позбавляє від можливості виникнення деяких з них.

Запропонований алгоритм проведення сеансу синхронізації дозволяє виявляти змінені дані на клієнті та на сервері з моменту проведення останньої синхронізації, зберігати ці зміни на обох вузлах та вирішувати конфлікти, які виникають в процесі збереження. Рішення базується на передачі замінених даних серверу та їх поширенню між всіма іншими вузлами.

Запропонована архітектура системи дозволяє здійснювати запити на проведення синхронізації даних з клієнтів на сервер та отримання всіх змін, що внесені до центральної бази даних з часу виконання останнього сеансу синхронізації, при використанні мінімальної кількості обчислювальних ресурсів. Особливістю розробки є можливість використання підходу на різних апаратних платформах. Описані ідеї можуть бути застосовані для розв'язання й інших наукових завдань.

## Список посилань


1. Отслеживание изменений в корпоративной базе данных SQL Server [Електронний ресурс]. Режим доступу: http://www.cyberguru.ru/database/sqlserver/tracking-changes.html.
2. Дейт К.Дж. Введение в системы баз данных, 6-е издание: Пер. с англ. / Дейт К.Дж. – К.: Издательский дом "Вильямс", 2000. – 848 с.
3. Третяк В.Ф. Тиражирование данных в системе управления базами данных / В.Ф. Третяк, Д.Ю. Голубничий, Ю.В. Челенко // Управління розвитком – Х.: ХНЕУ, 2005. – № 3. – С. 94-95.
4. Третяк В.Ф. Актуальність побудови grid систем / В.Ф. Третяк, В.М. Приходько, Д.Ю. Голубничий // Перша науково-технічна конференція Харківського університету Повітряних Сил. – Х.: ХУ ПС, 2005. – С. 240-241.
5. Третяк В.Ф. Технология репликации в распределенных системах управления базами данных / В.Ф. Третяк // Інформаційно-керуючі системи на залізничному транспорті. – Х.: ХАЗДТ, 2004. – № 1. – С. 7-10.
6. Третяк В.Ф. Проблеми розвитку програмного забезпечення середовища розподілених обчислень GRID / В.Ф. Третяк, Д.Ю. Голубничий, І.О. Золотарьова // Матеріали третьої наукової конференції Харківського університету Повітряних Сил мені Івана Кожедуба. – Х.: ХУПС, 2007. – С. 7.
7. Голубятников И.В., Матвеев Ю.В., Сергеев И.В. Создание программного комплекса репликации СУБД-независимых данных / И.В. Голубятников, Ю.В. Матвеев, И.В. Сергеев // Системный анализ, информатика и оптимизация. Сборник научных трудов. Под ред. д.т.н., проф. Букреев В.З. – М.: РЗИТЛП, 2001. – С. 106-116.
8. Синхронизация [Електронний ресурс]. Режим доступу: http://ru.wikipedia.org/wiki/Синхронизация_%28информатика%29.
9. Ciciani B. Analysis of Concurrency-Coherency Control Protocols for Distributed Transaction Processing Systems with Regional Locality / B. Ciciani, D.M. Dias, P.S. Yu // IEEE Transactions on Software Engineering. - October 1992. – Vol. 18, No. 10. – P. 899-914.
10. Конноли Т. Базы данных: проектирование, реализация и сопровождение. Теория и практика, 2-е изд.: Пер. с англ.: Уч. пос. / Т. Конноли. – М.: Издательский дом "Вильямс", 2000. – 1120 с.
11. Database Systems / B. Ciciani, D.M. Dias, P.S. Yu // IEEE Transactions on Knowledge and Data Engineering. – June 1990. – Vol. 2, No. 2. – P. 247-261.
12. Использование Oracle 8 / [Вильям Г. Пейдж и др.]. – К.; М.; СПб.: Изд. дом «Вильямс», 1998. – 752 с.
13. Мейер М. Теория реляционных баз данных / М. Мейер. – М.: Мир, 1987. – 608 с.

14. Саймон А.Р. Стратегические технологи баз данных: менеджмент на 2000 год. Пер. с англ. / А.Р. Саймон. Под ред. и с предисл. М.Р. Когаловского. – М.: Финансы и статистика, 1999. – 479 с.
15. В.Ф. Третьяк. Использование техрологии репликации в системе управления распределенными базами данных / В.Ф. Третьяк, И.Е. Кужель, В.М. Приходько // Збірник праць Хараківського університету Повітряних Сил, – Х., 2010. – № 2(24).
16. Киреев В.А. Проектирование хранилища данных социальной информации. – Конференция Сибирского федералного университета. Математика и компютерные науки, 2011.
17. А.Д. Плутенко, А.А. Ситников. Проблемы имитационного моделирования процесса журнальной репликации данных в распределенной СУБД. Амурский государственный университет 2005.
18. A. Tridgell, P. Mackerras. The Rsync Algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, 1996.