УДК 004.65: 004.9

Н. Я. Павич, Б. Є. Кутковий
Національний університет “Львівська політехніка”,
кафедра програмного забезпечення

СПОСІБ ПРИСКОРЕНОГО ОБСЛУГОВУВАННЯ API ЗАПИТІВ ДО СИСТЕМ УПРАВЛІННЯ ХМАРНИМИ БАЗАМИ ДАНИХ

Проаналізовано сучасний стан обслуговування Application Programming Interface (API) запитів до систем управління хмарними базами даних. Встановлено доцільність створення засобів щодо зменшення часу обслуговування таких запитів та ефективної синхронізованості локальної та хмарної баз даних. З’ясовано основні особливості та принципи реплікації даних. Обґрунтовано доцільність використання для реплікації даних лічильника поколінь замість системного таймера. Запропоновано асинхронний спосіб прискореного обслуговування API запитів до систем управління хмарними базами даних за рахунок застосування синхронізаційної акумулятивної таблиці та реєстрації змін у базах даних за допомогою двоетапного встановлення поколінь. Розроблено бібліотеку, яка забезпечує виконання асинхронних API запитів до системи управління хмарними базами даних Salesforce. Бібліотека може бути використана у будь-якому Ruby on Rails застосунку. Оцінено вигоди від запропонованих рішень на тестовому прикладі. Отримані результати тестових досліджень підтверджують мінімізацію часу обслуговування API викликів до систем управління хмарними базами даних за запропонованим асинхронним способом.

Ключові слова: API запити, прискорене обслуговування, системи управління базами даних, хмарна база даних.

N. Pavych, B. Kutkovyi
Lviv Polytechnic National University,
Department of Software

ACCELERATED SERVICING METHOD OF API CALLS TO CLOUD-DATABASE MANAGEMENT SYSTEMS

Analyzed the current state of the Application Programming Interface (API) calls to cloud database management systems. The expediency of creating tools to reduce the time for servicing such requests and the effective synchronization of the local and cloud databases has been established. The main features and principles of data replication are clarified. The expediency of using the counter of generations in the replication process instead of the system timer is justified. An asynchronous method of accelerated servicing for API calls to cloud database management systems is proposed by using a synchronization accumulative table and registering changes in databases using a two-stage set of generations. A library that provides the implementation of asynchronous API queries for the Salesforce cloud management system has been developed. The library can be used in any Ruby on Rails application. The evaluation of the benefits for the proposed solutions in the test case was carried out. The results of the test

studies confirm the minimization of the service time for API calls to the cloud database management systems based on the proposed asynchronous method.

Key words: API calls, accelerated servicing, database management systems, cloud database.

Вступ

Хмарні обчислення (англ. *Cloud computing*) – модель забезпечення зручного мережевого доступу на вимогу до деякого загального фонду конфігурованих обчислювальних ресурсів (наприклад, мереж передавання даних, серверів, пристроїв зберігання даних, додатків і сервісів – як разом, так і окремо), які можуть бути оперативно надані та звільнені з мінімальними експлуатаційними витратами або зверненнями до провайдера [1]. Технології хмарних обчислень є ефективними засобами для розподілених обчислень та істотно розширюють функціональні можливості комп'ютерних систем та мереж. Широке використання хмарних обчислень привело до того, що вони стали основою для багатьох інновацій та розв'язання складних обчислювальних задач. Вони поширені у багатьох комп'ютерних фірмах та у звичайних користувачів. Дослідники визнають, що хмарні технології є одним із перспективних напрямів розвитку обчислювальної техніки [1–3]. Відповідно розроблення нових ефективних засобів у царині хмарних технологій є актуальним завданням.

Аналіз останніх досліджень та публікацій

Хмарні технології передбачають використання для розв'язання прикладних задач різних апаратних та програмних засобів серверів мереж, методологій та інструментів, що надаються користувачеві як інтернет-сервіси для реалізації конкретних цілей, завдань, проектів.

Хмарні технології – це зручне середовище для зберігання і опрацювання інформації, яка об'єднує апаратні засоби, ліцензійне програмне забезпечення, канали зв'язку, а також технічну підтримку користувачів [4]. Хмарні обчислення – це певний базис-вектор, отриманий у результаті синтезу низки технологій і підходів [3]. Основними компонентами хмарних технологій можна вважати інфраструктуру, платформу, програмне забезпечення. Інфраструктура – це набір фізичних пристроїв (сервери, зовнішні запам'ятовувальні пристрої, канали передавання інформації). Платформа – набір послуг. Програмне забезпечення, що доступне за запитом користувачів [2, 5].

До основних функціональних можливостей хмарних технологій можна зарахувати:

- Доступ до особистої інформації з будь-якого комп'ютера, що підключений до Інтернету.
- Можливість працювати з інформацією з різних пристроїв (ПК, ноутбуки, планшети, телефони).
- Незалежність від операційної системи комп'ютера користувача – веб-сервіси працюють у браузері будь-яких ОС.
- Одну інформацію можна переглядати і редагувати одночасно з різних пристроїв.
- Багато платних програм є безкоштовними (або дешевими) веб-додатками.
- Запобігання втраті інформації, яка зберігається у хмарних сховищах.
- Завжди актуальна й оновлена інформація.
- Використання останніх версій програм і оновлень.
- Можливість об'єднання інформації з іншими користувачами
- Легко ділитися інформацією з людьми в будь-якій точці земної кулі.

Недоліками цих технологій можна вважати.

- Необхідність постійного з'єднання. Для отримання доступу до послуг "хмари" необхідне постійне з'єднання з мережею Інтернет.
- Обмеження у програмному забезпеченні. Є обмеження щодо ПЗ, яке можна розгортати на "хмарах" і надавати його користувачеві. Користувач має обмеження у використовуваному забезпеченні та іноді не має змоги налаштувати його під власні цілі.
- Конфіденційність. Конфіденційність даних, що зберігаються у публічних "хмарах", сьогодні викликає багато суперечок, але здебільшого експерти сходяться в тому, що не

рекомендується зберігати найцінніші для компанії документи на публічній “хмарі”, оскільки поки що немає технології, яка б гарантувала 100 % конфіденційність даних.

- Безпека. “Хмара” сама по собі є достатньо надійною системою, однак, проникнувши в неї, зловмисник отримує доступ до величезного сховища даних. Ще один мінус – використання систем віртуалізації, в яких використовуються ядра стандартних ОС (наприклад,Windows), що дозволяє проникати вірусам та призводить до вразливості системи.
- Дороге обладнання. Для побудови власної “хмари” необхідно виділити значні матеріальні ресурси, що невигідно щойно створеним і малим компаніям.
- Подальша монетизація ресурсу. Цілком можливо, що компанії надалі вирішать збільшувати плату користувачів за надані послуги.

Хмарні бази даних – це бази даних, які запускаються на платформах хмарних обчислень, таких як Amazon EC2, GoGrid і Rackspace. Існують дві найпоширеніші моделі розгортання: користувачі можуть придбати безпосередньо послугу доступу до баз даних, які обслуговує постачальник хмарного сервісу, або ж запустити бази даних у “хмарі” незалежно, використовуючи образ віртуальної машини. Серед хмарних баз даних є як SQL-орієнтовані, так і інші, що використовують модель даних NoSQL [6].

Використовують декілька підходів для запуску та управління базами даних у “хмарі”.

- Образ віртуальної машини – хмарні платформи дозволяють користувачам отримати екземпляр віртуальної машини на обмежений час, і користувач може запустити базу даних на цій віртуальній машині. Користувачі можуть або самостійно завантажувати образ із встановленою на ньому базою даних, або використовувати попередньо створені образи, що вже містять оптимізовану інсталяцію. Наприклад, Oracle забезпечує готовий машинний образ з інсталяцією Oracle Database 11g Enterprise Edition на Amazon EC2 та на Microsoft Azure [7, 8].
- База даних як сервіс (DBaaS) – деякі хмарні платформи надають опції для використання бази даних як сервісу, без фізичного запуску екземпляра віртуальної машини для бази даних. За такої конфігурації власникам додатків не потрібно встановлювати та підтримувати базу даних. Натомість провайдер сервісу бази даних бере на себе відповідальність за її встановлення та підтримку, і власники додатків платять згідно з їх використанням. Наприклад, Amazon Web Services надає три сервіси баз даних як частину власної хмарної пропозиції: Amazon SimpleDB, NoSQL сховище ключ-пара; Amazon Relational Database Service, сервіс бази даних на SQL з MySQL інтерфейсом; та Amazon DynamoDB. Подібно Microsoft надає Azure SQL Database сервіс як частину їх хмарної пропозиції [8].
- Інша опція містить керований хостинг бази даних у “хмарі”, де постачальник хмарної БД не пропонує базу даних як сервіс, а хостить базу даних і керує нею від імені власника додатка [8].

Особливості управління базами даних у “хмарі” такі. Більшість сервісів БД надають веб-орієнтовані консолі, що кінцевий користувач може використовувати для підготовки та налаштування екземплярів БД. Наприклад, Amazon Web Services веб-консоль дає змогу користувачам запускати екземпляри БД, створювати снепшоти (схоже до бекапів) баз даних, і спостерігати за статистикою БД [8].

Сервіси баз даних складаються із компоненти-менеджера БД, що керує основними екземплярами БД, використовуючи прикладний програмний інтерфейс (Application Programming Interface-API) цього сервіса. API сервіси розкриті для кінцевого користувача та дають йому змогу виконувати обслуговування та операції зміни розміру на власних екземплярах БД. Наприклад, Amazon Relational Database Service’s сервісне API дозволяє створювати екземпляр БД, здійснювати модифікацію ресурсів, доступних для екземпляра БД, видаляти екземпляр БД, створювати снепшоти (схожі до бекапів) баз даних та відновлювати БД зі снепшоту [8].

Сучасний підхід до використання API сервісу систем управління хмарними базами даних недостатньо опрацьований. Проблема полягає в тому, що API виклики можуть тривати упродовж доволі довгих періодів часу і недостатньо досліджена технологія реплікації даних. В багатьох випадках це згубно позначається на результатах виконання користувачами прикладних завдань.

Зазначений недолік можна усунути прискореним обслуговуванням API запитів до систем управління хмарними базами даних.

Мета статті

Мета досліджень – пошук способу прискореного обслуговування API запитів до систем управління хмарними базами даних.

Основні результати дослідження

Прискорене обслуговування API запитів до систем управління хмарними базами даних (ХБД) є найперспективнішим за рахунок раціонального відстеження змін у локальній базі даних, синхронізації їх на хмарній базі даних, або навпаки. Одним із поширених механізмів синхронізації вмісту декількох копій об'єкта (наприклад, вмісту бази даних) є реплікація (англ. Replication) [9–11]. Під час реплікації зміни, зроблені в одній копії об'єкта, можуть бути поширені в інші копії. Розрізняють синхронні та асинхронні реплікації.

У разі синхронної реплікації, якщо якась репліка оновлюється, всі інші репліки того ж фрагмента даних також повинні бути оновлені в тій самій транзакції. Логічно це означає, що існує лише одна версія даних [9]. У більшості продуктів синхронна реплікація реалізується за допомогою тригерних процедур (можливо, прихованих і керованих системою). Але синхронна реплікація має той недолік, що створює істотне додаткове навантаження під час виконання всіх транзакцій, в яких оновлюються будь-які репліки (крім того, можуть виникати проблеми, пов'язані з доступністю даних).

У разі асинхронної реплікації оновлення однієї репліки поширюється на інші через деякий час, а не в тій самій транзакції. Отже, за асинхронної реплікації вводиться затримка або час очікування, протягом якого окремі репліки можуть бути фактично неідентичними (тобто визначення "репліка" виявляється не зовсім відповідним, оскільки ми не маємо справу з точними і своєчасно створеними копіями) [9].

На підставі аналізу наведених особливостей можна встановити, що прискорене обслуговування API запитів до систем управління ХБД досягається асинхронними реплікаціями. Тому доцільно розглянути відповідний алгоритм.

Одним із основних елементів асинхронної реплікації є первинний ключ, який переважно запакований 64-розрядним двійковим кодом (NUMERIC (18,0)). Старші чотири байти – ID ЛБД, молодші чотири байти – ID записи. Фактично це – складовий ключ, записаний в одне поле. Є кілька рішень, що забезпечують унікальність первинного ключа. Популярним рішенням є методи, основані на перетворенні ключів (один і той самий запис має різний первинний ключ у різних ЛБД). Однак перетворення ускладнює алгоритм і вимагає додаткового часу.

Найпростіше рішення – використовувати для реєстрації змін часові мітки (time stamp). Але у часових міток є безліч недоліків:

- дуже важко встановити однаковий час на всіх вузлах ХБД;
- системний час залежить від людського фактора: його можна виставити як завгодно;
- різноманітні системи часу: UTC, поясний час тощо;
- різноманітні способи подання часу в комп'ютері; у кожної OS, СУБД, мови програмування може бути своє уявлення часу, з різною точністю і обмеженнями.

З усього цього випливає, що time stamp краще за можливості не використовувати. Замість time stamp застосовується аналог логічного таймера (logical clock), який називається лічильником поколінь. Значення logical clock збільшується після кожної події (в цьому випадку подія – зміна), отже, можна відновити черговість змін в одній ЛБД, але не можна відновити черговість змін у ХБД, тому що не можна порівнювати значення logical clock у різних ЛБД. Лічильник поколінь відрізняється від logical clock тим, що покоління збільшуються в загальному випадку після декількох змін. Тоді ці зміни належать одному поколінню.

Для реєстрації змін в алгоритмі використовують службові таблиці, т. зв. таблиці реєстрації (логи), які мають такі властивості.

− кожній таблиці XXXX (XXXX – відповідна таблиця даних) з даними відповідає одна таблиця реєстрації LOG_XXXX;

− кожному запису в таблиці реєстрації відповідає один запис у таблиці даних, якщо запис вставлявся або змінювався;

− запису в таблиці реєстрації не відповідає жоден запис у таблиці даних, якщо запис видалено.

ID записи в таблиці даних є зовнішнім ключем, який посилається на відповідний запис у таблиці реєстрації. Таблиця реєстрації LOG_XXXX має таку структуру (табл. 1).

Таблиця 1

Структура таблиці реєстрації

Призначення	Фізичне ім’я	Примітка
ID запису	LOG_GID	
Покоління вставки	LOG_INS_GEN	
Покоління зміни/видалення	LOG_UPD_GEN	
Ознака видалення	LOG_IS_DELETED	Якщо немає підпорядкованого запису в таблиці даних, значить, ознака віддаленості дорівнює True (1), інакше – False (0)
ID ЛБД-власника запису (ЛБД, в якій запис створено)	LOG_DB_ID	Обчислюється за ID записом LOG_GID.

Загальні особливості алгоритму реплікації. БД, яка отримує дані, назвемо клієнтом. БД, з якої дані зчитуються, назвемо сервером. Реплікація починається з ініціативи клієнта. Послідовність передавання даних можна розділити на такі основні кроки.

1. Запросити поточний стан лічильника поколінь сервера (Server.CURR_GEN).

2. Для кожного покоління, починаючи від відомого покоління сервера + 1 (Вибрати з DB_PROFILE) до Server.CURR_GEN (зміни невідомі клієнту), виконати:

2.1. Стартувати транзакцію; встановити стан реплікації (Client.BEGIN_RECEIVE).

2.2. Прийняти зміни із сервера (Server.DB_ID_GEN, Server.MASTER_GEN, Server.DETAIL_GEN); зміни запам’ятовують у буферних таблицях (Client.DB_PROFILE_INPUT, Client.MASTER_INPUT, Client.DETAIL_INPUT).

2.3. Застосувати зміни (Client.PROCESS_INPUT_TABLES): оновлення (порядок головна–підлегла); вставка (порядок головна–підлегла); видалення (порядок підпорядкована–головна).

2.4. Відключити стан реплікації, запам’ятати покоління сервера (Client.END_RECEIVE); підтвердити транзакцію.

2.5. Встановити нове покоління (Client.SET_NEW_GEN); встановлення покоління виконується в спеціальній транзакції.

У разі виняткової ситуації реплікацію можна продовжити з останнього успішно прийнятого покоління, а не спочатку. Під час реалізації сервер запам’ятовує, яке покоління прийняв клієнт (Server.SET_CLIENT_GEN). Це потрібно для очищення логів від мертвих записів. За винятком виклику Server.SET_CLIENT_GEN дані передаються тільки від сервера клієнту.

Прискорене обслуговування API запитів до систем управління хмарними базами даних (ХБД) можна здійснити на основі видозміни алгоритму, забезпечивши фонову синхронізації даних ЛБД та системою управління ХБД. Принципи алгоритмічно-програмної реалізації цього підходу показано на прикладі здійснення різнобічної інтеграції даних між системою управління хмарними базами даних Salesforce та Ruby on Rails застосунком. Для практичності використання програмний засіб доцільно подати у вигляді gem’у (пакет з бібліотекою) [10].

За результатами досліджень реалізовано програмний продукт з такими основними функціями. На основі конфігураційного файла з чітко визначеними координатами кінцевих точок система може створювати фоновий канал зв'язку між локальною базою даних, Ruby on Rails додатком та хмарною базою даних на платформі Salesforce. Можливе задання різних видів стратегій синхронізації (постійної, пасивної, асоціативної), синхронізація окремих полів та асоціативних зв'язків у базі даних. Описуючи логіку синхронізації в реляційних моделях, користувач має змогу вказати окремі поля в таблиці для синхронізації, а також вид асоціативного зв'язку з іншими таблицями в базі даних. Система аналізує опис та здійснює синхронізацію асоціативних таблиць й окремих полів. Всі події синхронізації відбуваються асинхронно, в окремому потоці, що забезпечує набагато вищий рівень обслуговування порівняно з прямими API викликами.

Для розроблення тестового програмного забезпечення щодо синхронізації даних між локальною базою даних та хмарною базою даних на платформі Salesforce використано мову програмування Ruby версії 2.2.1. Як програмний каркас – Rails, версії 4.2.7. Для написання вихідного коду програми – Rubymine 2017, а для зберігання даних – PostgreSQL9.5. Для універсальності застосування у будь-якому Ruby on Rails додатка програмне забезпечення реалізовано у формі бібліотеки (gem). Використано менеджер пакетів RubyGems для спрощення процесу створення, поширення і встановлення бібліотек (розподілений пакетний менеджер) [10,11].

Логічні умови режимів функціонування бібліотеки встановлено такими модулями.

1. Restforce::DB::Associations::Base – встановлює асоціативний зв'язок між двома мапінгами в реєстрі мапінгів.
2. Restforce::DB::Associations::BelongsTo – визначає зв'язок належності рекорду до певної таблиці в SalesForce базі даних.
3. Restforce::DB::Associations::ForeignKey – визначає salesforce_id як зовнішній ключ.
4. Restforce::DB::Associations::HasMany – визначає зв'язок належності багатьох SalesForce рекордів до одного SalesForce рекорду.
5. Restforce::DB::Associations::HasOne – визначає зв'язок належності одного SalesForce рекорду до певної таблиці в базі даних SalesForce.
6. Restforce::DB::Instances::Salesforce – слугує обгорткою для об'єктів SalesForce, надаючи спільний API для врегулювання атрибутів рекорду з екземплярами ActiveRecord.
7. Restforce::DB::Middleware::StoreRequestBody – забезпечує зберігання тіла запиту в енвайренменті за допомогою HTTP-клієнта Faraday.
8. Restforce::DB::RecordTypes::Salesforce – слугує обгорткою для єдиного SalesForce об'єкта класу, уможливлює огляд його атрибутів та мапінг атрибутів.
9. Restforce::DB::Strategies::Always – визначає стратегію ініціювання для мапінгу, в якій новознайдені рекорди повинні бути завжди синхронізовані з Salesforce в базу даних та навпаки.
10. Restforce::DB::Strategies::Associated – визначає стратегію ініціювання для мапінгу, в якій новознайдені рекорди повинні бути синхронізовані в інший системі, але лише в тому випадку, коли певний асоціативний рекорд буде синхронізовано.
11. Restforce::DB::Strategies::Passive – визначає стратегію ініціювання для мапінгу, в якій новознайдені рекорди не повинні бути синхронізовані в іншій системі.
12. Restfoce::DB::Worker – являє собою постійний цикл опитування, через який відбувається вся синхронізація рекордів.
13. Restforce::DB::Synchronizer – відповідає за синхронізацію рекордів у Salesforce з рекордами в локальній базі даних.
14. Restforce::DB::Registry – відповідає за стеженням всіх мапінгів, що потрапляють до його реєстру.
15. Restforce::DB::Railtie – надає можливість запускати необхідні Rake:Tasks в будь-якому Rails застосунку.
16. Restforce::DB::Mapping – захоплює безліч відображень між стовпцями бази даних і полем Salesforce, забезпечуючи утиліти для перетворення хеш-атрибутів від одного до іншого.
17. Restforce::DB::DSL – визначає синтаксис, за допомогою якого відображення може бути налаштоване між моделлю бази даних і типом об'єкта в Salesforce.

18. Restforce::DB::Configuration – надає методи прямолінійного читання і запису, щоб дати користувачам змогу налаштовувати Restforce :: DB.

19. Restforce::DB::Accumulator – відповідає за накопичення змін протягом одного прогону синхронізації.

Підхід щодо мінімізації часу обслуговування API викликів, який реалізовується створюваною бібліотекою, передбачає використання локального сховища на основі будь-якої реляційної бази даних, використання ORM [12], що спрощує читання та модифікування даних у таблицях бази даних та синхронізацію між віддаленою хмарною базою даних загалом. Також цей підхід передбачає в себе фонове опрацювання даних без втручання користувача у синхронізацію.

Базовими операціями, які використовують більшість систем управління хмарними базами даних, є CRUD (create, read, update, delete) операції [13–15]. Ці операції інтерпретуються у відповідні запити, які віддалена база даних використовує для безпосередньої роботи.

Спосіб прискореного обслуговування API запитів до систем управління хмарними базами даних ґрунтується на застосуванні синхронізаційної акумулятивної таблиці. Первинним етапом є формування цієї таблиці, що міститиме всю необхідну інформацію про те, які саме зміни відбулись у локальній та віддаленій базах даних з моменту останньої синхронізації. На основі аналізу цієї таблиці роблять висновок про те, які саме із записів потребують оновлення, вставляння або ж видалення як у локальній, так і у віддаленій базах. Відповідно засоби синхронізації повинні складатися з трьох основних частин – формувача акумулятивної таблиці, аналізатора таблиці та модуля для виконання API викликів. Ці частини функціонально незалежні одна від одної, тому під час подальших модифікацій можна замінити кожну з них з метою покращення результатів, і це не вплине на роботу іншого модуля. Частини доцільно реалізовувати як модулі. Це даватиме змогу використати їх у різних застосунках – веб-сайтах, десктопних або навіть мобільних додатках. Своєю чергою, кожен з модулів містить декілька підмодулів. Загальну структуру основних засобів синхронізації наведено на рис. 1.

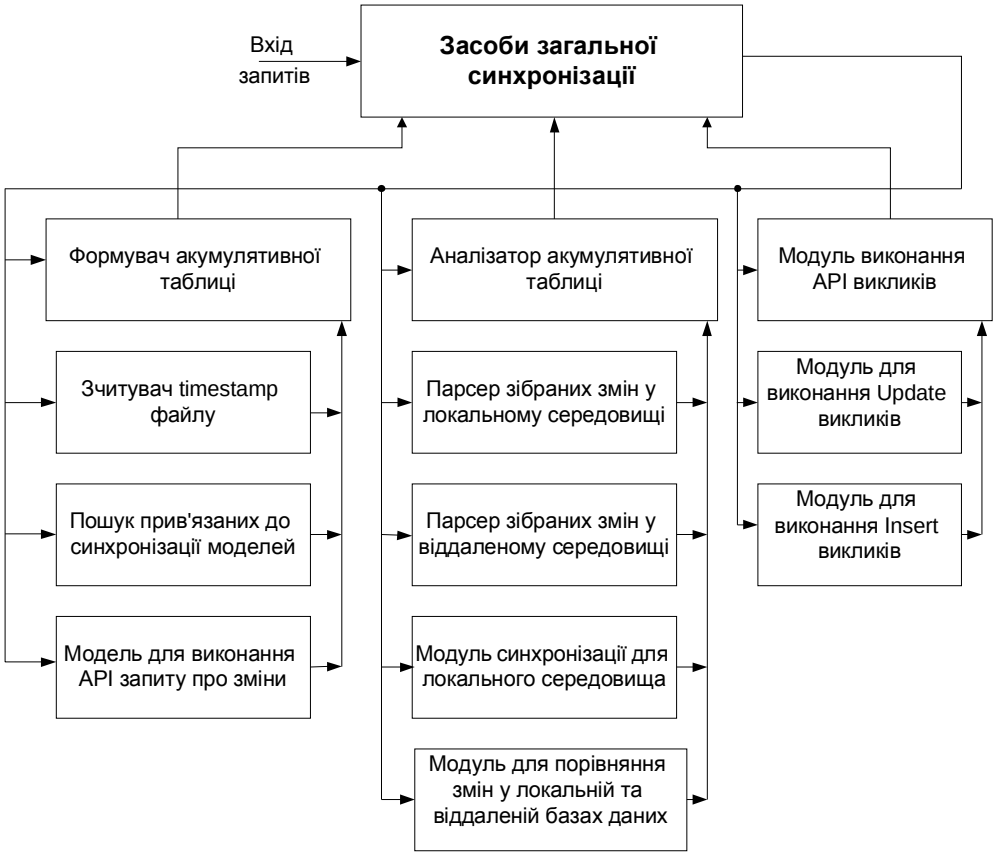

Рис. 1. Загальна структура компонентів засобів синхронізації

Бібліотеку модулів щодо мінімізації часу обслуговування API викликів можна впровадити у будь-який Ruby on Rails застосунок. Розробник має можливість прив'язати будь-яку модель (з умовою, що модель відповідає вигляду віддаленої таблиці) до синхронізації з віддаленою таблицею у хмарній базі даних.

Адміністратор, що адмініструє систему управління хмарними базами даних, має можливість модифікувати дані віддалених таблиць. Як результат таблиця в локальній базі даних повинна набути такого ж стану, як і віддалена.

Алгоритм роботи застосунку можна умовно поділити на декілька лінійних та нелінійних сценаріїв. Узагальнену схему алгоритму наведено на рис. 2.

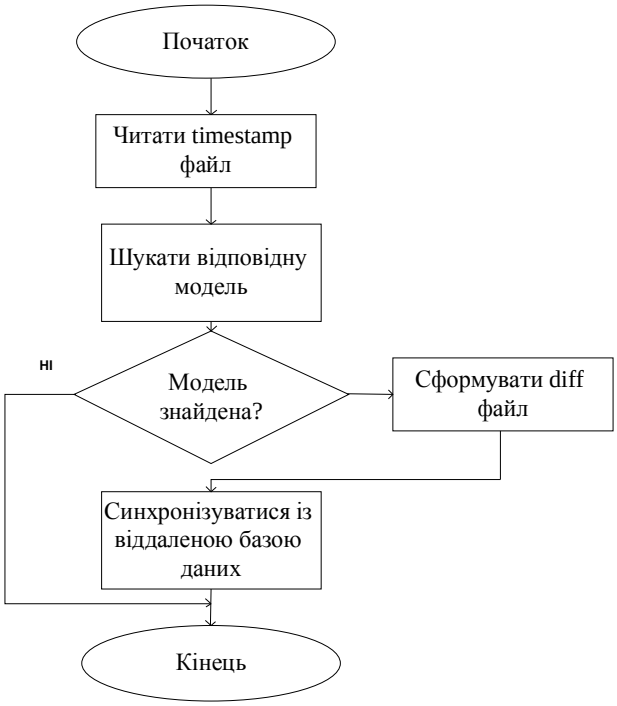

Рис. 2. Узагальнена схема алгоритму роботи застосунку

Робота застосунку подається через відповідний цикл, ітерація якого здійснює синхронізацію даних. Цикл відбувається у фоновому режимі без прямого спілкування з користувачем.

На ітерації кожного циклу відбуваються такі процеси:

1. Читання timestamp файла. В цьому файлі містяться дані, що вказують, коли останній раз відбулась успішна синхронізація.
2. Пошук під'єднаних моделей. За допомогою спеціальних DSL [13–15] вказують, які з моделей ActiveRecord ORM та їх полів підлягають синхронізації.
3. Формування diff файла на основі змін у записах в локальній та віддаленій базі даних та timestamp даних.
4. Двобічна синхронізація.

Процес синхронізації даних полягає у поетапному розгляді diff файла та виконанні insert, update, delete дій стосовно локальної або ж до віддаленої бази даних.

Diff файл генерується на основі змін, що відбулись на локальній та віддаленій базах даних. Тобто умовно його можна поділити на дві секції – local_diff та remote_diff. У першій секції містяться дані про те, які зміни відбулись у локальній базі даних, у другій – у віддаленій. Важливим фактом є те, що видалення з локальної бази даних автоматично вноситься до local_diff. Інакше за допомогою timestamp ці зміни неможливо було б відстежити.

Деталізований процес синхронізації даних подано через схему алгоритму, що зображена на рис 3. У результаті локальна та віддалена бази даних набувають синхронізованого вигляду.

Важливими є такі методи цього алгоритму. Collector.Run – метод run запускає процес поетапного збирання даних, які були модифіковані, як у віддаленій базі даних, так і в локальній. TimestampCache.timestamp – отримує найновіший збережений timestamp для переданого об’єкта. Timestamp потрапляє до відмічених часових міток, щоб переконатися, що цей цикл знає про модифікації, здійснені протягом попереднього пробігу. Synchronizer.run – синхронізує записи для поточного порівняння між змінами в локальній та віддаленій базах даних. Він покладається на з’єднання, які є у моделях, що підлягають синхронізації.

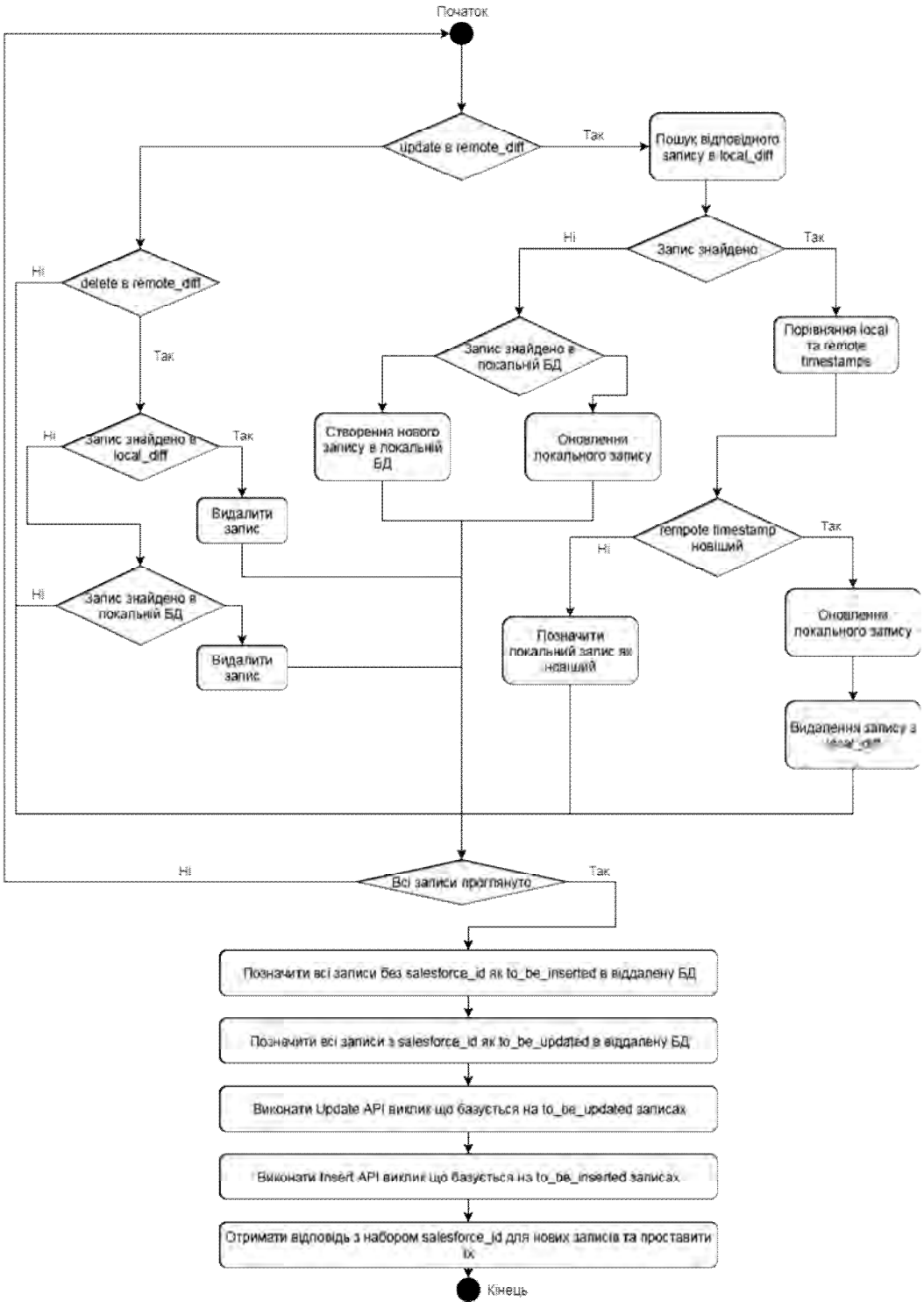

Рис. 3. Деталізована схема алгоритму роботи застосунку

Оцінку вигоди стосовно застосування способу прискореного обслуговування API запитів здійснено на основі тестових досліджень із використанням десяти типових запитів до систем управління хмарними базами даних. Типові запити отримано за допомогою інструменту Active Record ORM [10, 16]. Для якісної оцінки запропонованих покращень порівняно фактичний час виконання запитів на однаковому наборі даних за допомогою синхронних та асинхронних викликів. Розраховано кількісні значення абсолютної та відносної вигоди у швидкодії.

Після запуску бібліотеки, використовуючи демонстраційний застосунок, порівняли швидкодію звичайних та асинхронних API викликів. У цих порівняннях використано базові операції, що виконуються в будь-якій хмарній або ж локальній базах даних. Порівняння швидкодії здійснено для операцій: Select – оператор для реалізації вибірки даних; Update – оператор для реалізації модифікації даних; Insert – оператор для реалізації вставляння даних; Delete – оператор для видалення даних. Для кожної операції виконано десять типових API викликів. Результати наведено в табл. 2 та 3. Час обслуговування подано у секундах.

Таблиця 2

Час обслуговування API запитів для операторів Select та Update

SELECT			UPDATE		
API type	Sync	Async	API type	Sync	Async
1	1,5123	0,012392	1	2,353	0,06832
2	1,32342	0,074335	2	2,6438	0,03715
3	1,74356	0,020029	3	2,3843	0,07452
4	2,00016	0,087364	4	2,0123	0,05621
5	1,7722	0,024306	5	1,9982	0,09013
6	1,86553	0,03413	6	2,743	0,05232
7	1,78625	0,041127	7	2,6285	0,06016
8	1,7971	0,027842	8	2,8478	0,01901
9	1,24168	0,017419	9	2,348	0,07727
10	1,54275	0,064045	10	2,4395	0,02872
Min time, s	1,24168	0,012392	Min time, s	1,9982	0,01901
Speedup	1	100,2001	Speedup	1	105,113
Efficiency	1	100,2001	Efficiency	1	105,113

Таблиця 3

Час обслуговування API запитів для операторів Insert та Delete

INSERT			DELETE		
API type	Sync	Async	API type	Sync	Async
1	2,78079	0,04799	1	2,29231	0,02165
2	2,28276	0,02063	2	2,13412	0,02257
3	2,71386	0,01444	3	2,20325	0,02239
4	2,24497	0,02318	4	2,24966	0,02613
5	2,67617	0,03801	5	2,19296	0,03841
6	2,40364	0,06328	6	2,11411	0,03855
7	2,89672	0,01639	7	2,12011	0,03754
8	2,29987	0,03267	8	2,13678	0,03628
9	2,50797	0,02989	9	2,27608	0,03497
10	2,81776	0,03140	10	2,17742	0,03919
Min time, s	2,24497	0,01444	Min time, s	2,11411	0,02165
Speedup	1	155,427	Speedup	1	97,6246
Efficiency	1	155,427	Efficiency	1	97,624600

Для кожної базової операції значення часу обслуговування типових API запитів приведено до усереднених значень та подано у табл. 4.

Таблиця 4

Порівняння усередненого часу обслуговування API викликів для базових операцій

Оператор	Час обслуговування звичайних API викликів, sec	Час обслуговування API викликів асинхронним способом, sec
SELECT	1,52	0,02
UPDATE	3,57	0,039
INSERT	2,21	0,032
DELETE	2,71	0,0211

Отримані результати тестових досліджень підтверджують мінімізацію часу обслуговування API викликів до систем управління хмарними базами даних за запропонованим асинхронним способом.

Висновки

Аналіз сучасного стану обслуговування API запитів до систем управління хмарними базами даних показав доцільність створення засобів щодо зменшення часу обслуговування таких запитів та ефективної синхронізованості локальної та хмарної баз даних. З'ясовано основні особливості та принципи реплікації даних. Обґрунтовано доцільність використання для реплікації даних лічильника поколінь замість системного таймера. Запропонований асинхронний спосіб прискореного обслуговування API запитів до систем управління хмарними базами даних оснований на застосуванні синхронізаційної акумулятивної таблиці та реєстрації змін у базах даних за допомогою двоетапного встановлення поколінь. Розроблена програмна бібліотека забезпечує виконання асинхронних API запитів до системи управління хмарними базами даних Salesforce. Бібліотека може бути використана у будь-якому Ruby on Rails застосунку. Оцінка вигоди від запропонованих рішень основана на тестовому прикладі. Отримані результатами тестових досліджень підтверджують мінімізацію часу обслуговування API викликів до систем управління хмарними базами даних за запропонованим асинхронним способом.

1. Chappell D. A Short Introduction to Cloud Platforms an Enterprise–Oriented View: Chappell and Associates, San Francisco, 2008, pp. 1–13. 2. Jon-David Chappell & Associates, 2008. – P. 3-4es, M. Tim, Cloud Computing with Linux / Jones, Jones, M. Tim – IBM DeveloperWorks (2008-09-10). 3. Gillam, Lee. Cloud Computing: Principles, Systems and Applications / Nick Antonopoulos, Lee Gillam. – L. : Springer, 2010. – 23–24 p. 4. SoCC '10: Proceedings of the 1st ACM symposium on Cloud computing / Hellerstein, Joseph M. – N. : ACM, 2010. – 2 p. 5. Hassan, Qusay Demystifying Cloud Computing / Hassan, Qusay // The Journal of Defense Software Engineering. CrossTalk, 2011. – 16–21 p. 6. Peter Mell and Timothy Grance The NIST Definition of Cloud Computing / Peter Mell and Timothy Grance. – National Institute of Standards and Technology: U. S. Department of Commerce. doi:10.6028/NIST.SP.800-145. Special publication, 2011. – 32–35 p. 7. Baburajan, Rajani. The Rising Cloud Storage Market Opportunity Strengthens Vendors / Baburajan, Rajani. It.tmcnet.com [web resource] : It.tmcnet.com. 8. Gruman, Galen. What cloud computing really means. – [Web resource]: https://en.wikipedia.org/wiki/InfoWorld. 9. Antonio Regalado "Who Coined 'Cloud Computing?– Technology Review. MIT., 2013. – 30 p. 10. David Hansson. Ruby on Rails will ship with OS X 10.5 (Leopard) / [Web resource]. – http://weblog.rubyonrails.org/2006/8/7/ruby-on-rails-will-ship-with-os-x-10. 11. Martin Fowler. Patterns of enterprise application architecture. Addison-Wesley. 2015. – 47 p. 12. Steven Feuerstein, Bill Pribyl. Oracle PL/SQL Programming. 18.5 Modifying Persistent Objects. Retrieved 23 August 2011. – 296 p. 13. Mernik M., Heering J., Sloane A. M. When and how to develop domain-specific languages. ACM Computing Surveys, 2005. 316–317 p. 14. Rassokhin A.; Oleksyuk D. TDSS botnet: full disclosure. Retrieved 6 December 2012. 25 p. 15. Stonebraker, M. Rowe, LA. The POSTGRES data model (PDF). Proceedings of the 13th International Conference on Very Large Data Bases. Brighton, England: Morgan Kaufmann Publishers. 2016. 83–96 p. 16. Kutkovy B., Pavych N. API-calls optimization for cloud database management systems // International Scientific Journal "Internauka". – 2017, No. # 14. – [web resource]: https://www.inter-nauka.com/en/issues/2017/14/3003