УДК 004.065 **DOI:10.30837/rt.2022.4.211.03**

В.І. ЄСІН, д-р техн. наук, В.В. ВІЛІГУРА

ОСНОВНІ КАТЕГОРІЇ NEWSQL БАЗ ДАНИХ ТА ЇХ ОСОБЛИВОСТІ

Вступ

У сучасному світі все гостріше постає проблема роботи з величезними обсягами даних та великими навантаженнями. Великі Web-застосунки, соціальні мережі, торгові платформи, портали новин, різні наукові дослідження, бізнес-аналітика, а також безліч інших областей, так чи інакше, стикаються з проблемами керування та аналізу даних великого обсягу. У таких проєктах велика кількість користувачів одночасно читають та записують інформацію, що вимагає від системи керування даними не тільки великої пропускної здатності та низьких затримок, а й масштабованості, надійності та узгодженості даних. Донедавна реляційні системи керування даними (СКБД) залишалися головним інструментом керування даними, яким потрібно щодня обробляти мільйони запитів. За деякими оцінками експертів, ринок реляційних баз даних (БД) приносить дохід понад 50 мільярдів доларів [1]. Таку популярність їм забезпечила декларативна мова запитів SQL, за допомогою якої користувачі можуть оперувати даними. Однак, незважаючи на велику популярність, досвід застосування, універсальність, забезпечення безпеки даних, що зберігаються, традиційні реляційні СКБД не завжди можуть задовольнити вимоги сучасних застосунків. Соціальні мережі, наприклад, вимагають виконання мільйонів операцій зчитування та мільярдів операцій запису в режимі, близькому до реального часу [2]. Для того щоб впоратися зі зростаючим обсягом даних і трафіку, були потрібні більші обчислювальні ресурси. Для вирішення цієї проблеми відомі два підходи – так звані вертикальне (scaling up або scaling vertically) та горизонтальне (scaling horizontally або scaling out) масштабування.

Багато хто пробував найбільш очевидний варіант вертикального масштабування своїх СКБД, переміщуючи базу даних на машину з більш досконалим та потужнішим (продуктивним) обладнанням. Однак це тільки підвищує продуктивність, але, як правило, не має такої ефективної віддачі (більші машини стають дедалі дорожчими), не кажучи вже про те, що існує фізична межа критичних характеристик відповідного обладнання. Крім того, переміщення бази даних з однієї машини на іншу – складний процес, що часто потребує значного часу простою, що є неприйнятним для певних застосунків. Альтернативою було використання великої кількості невеликих машин, об'єднаних у кластер (горизонтальне масштабування). Кластер невеликих машин може використовувати звичайне апаратне забезпечення і в результаті здешевити масштабування. Крім того, кластер надійніший – окремі машини можуть вийти з ладу, але весь кластер продовжить функціонування, незважаючи на ці збої, забезпечуючи високу надійність.

Коли відбулося зрушення у бік кластерів, виникла нова проблема – реляційні бази даних не були призначені до роботи на кластерах. Кластерні реляційні бази даних, такі як Oracle RAC або Microsoft SQL Server, ґрунтуються на концепції загальної дискової підсистеми. Вони використовують кластерну файлову систему, яка виконує запис даних у легко доступну дискову підсистему, але це означає, що дискова підсистема, як і раніше, є єдиним джерелом вразливості кластера. При цьому в роботі [3] зазначається, що все ж таки реляційні бази даних можуть працювати на різних серверах з різними наборами даних, ефективно виконуючи фрагментацію (sharding) бази даних. Хоча це дозволяє розділити робоче навантаження, вся процедура фрагментації, як правило, повинна контролюватись застосунком, який повинен стежити за тим, який сервер бази даних і за якою частиною даних звертається. Втім, сьогодні, наприклад, у СКБД Oracle Database використовується функція Oracle Sharding, яка дозволяє автоматично розподіляти та реплікувати дані в пулі баз даних Oracle, які не використовують спільно апаратне або програмне забезпечення [4].

Тому деякі компанії почали створювати спеціальне програмне забезпечення (ПЗ) проміжного рівня (або проміжне ПЗ) для поділу СКБД з одним вузлом – на кластер менш дорогих машин. Таке проміжне програмне забезпечення надає застосунку єдину логічну базу даних, що зберігається на декількох фізичних вузлах (серверах). Коли застосунок видає запити до такої бази даних, проміжне ПЗ перенаправляє та/або перезаписує їх, щоб розподілити їх виконання на одному або кількох вузлах у кластері. Вузли виконують ці запити та надсилають результати назад проміжному програмному забезпеченню, яке потім поєднує їх в одну відповідь застосунку. Як відомі приклади такого проміжного програмного забезпечення є кластер eBay на базі Oracle, кластер Google на базі MySQL, Facebook для їх власного кластера MySQL [5, 6]. Однак слід враховувати, що таке ПЗ проміжного рівня добре підходить для простих операцій, таких як читання або оновлення одного запису, але не для запитів, які оновлюють більше одного запису в транзакції або в таблицях з'єднання (їх виконати набагато складніше). Згодом деякі компанії відмовилися від проміжного програмного забезпечення та почали розробляти власні розподілені СКБД. Така їх поведінка була обумовлена кількома причинами. Насамперед, традиційні СКБД того часу були орієнтовані на узгодженість/несуперечність (consistency) та коректність (correctness) за рахунок доступності та продуктивності. Але цей компроміс вважався неприйнятним для застосунків, яким необхідно постійно перебувати в мережі та підтримувати велику кількість одночасних операцій. По-друге, вважалося, що використання повнофункціональної СКБД, такої, наприклад, як MySQL, як звичайне сховище даних вимагає надто великих накладних витрат. Крім того, так само вважалося, що реляційна модель є не найкращим способом представлення даних, а використання SQL – це надлишок для простих пошукових запитів. Тому, як відомо, до кінця 2000-х років з'явився різноманітний набір масштабованих і доступніших розподілених NoSQL СКБД. Основними причинами використання можливостей технології NoSQL можна вважати [3]: необхідність забезпечити доступ до даних, обсяг яких та вимоги до продуктивності змушують використовувати кластери; необхідність підвищення продуктивності розробки застосунків за рахунок використання зручнішого способу взаємодії з даними. Кожне рішення в рамках технології NoSQL використовує свою модель. Ці моделі можна розділити на кілька категорій [3, 7]: сімейство стовпців/сховище широких стовпців (column families / wide column store); сховища документів/документні бази даних (document store); ключ-значення (key value/tuple store); графові (graph); багатомодельні (multimodel); об'єктні/об'єктно-орієнтовані (object / object oriented); XML; багатозначні (multivalue); подієві (event, event stores, event sourcing); тимчасові ряди (time series) та деякі інші. Перевага використання системи NoSQL (принаймні така достатньо поширена думка існувала) полягала в тому, що розробники могли зосередитися на тих аспектах свого застосування, які були більш корисні для їхнього бізнесу, замість того, щоб турбуватися про те, як масштабувати СКБД. Однак, незважаючи на те, що всі вони спрямовані на вирішення проблеми масштабованості та мають певні переваги, кожному з них притаманні різні недоліки і кожне рішення може ефективно справлятися лише з певним класом завдань. Багато застосунків не можуть використовувати системи NoSQL, оскільки вони не можуть відмовитися від жорстких вимог, що висуваються до транзакцій та узгодженості. При цьому деякі організації, перш за все Google, виявили, що СКБД NoSQL змушують їх розробників витрачати занадто багато часу на написання коду для обробки суперечливих даних. Наприклад, решардинг (resharding) однією з систем зайняв понад два роки інтенсивних зусиль і включав координацію та тестування десятків команд для мінімізації ризиків, тоді як використання транзакцій робить їх більш продуктивними, оскільки вони являють собою більш зрозумілу для обговорення абстракцію [8]. Таким чином, єдиним доступним для цих організацій варіантом було або придбання потужнішої одновузлової машини і вертикальне масштабування СКБД, або розробка власного проміжного програмного забезпечення, що підтримує транзакції сегментованої БД. Обидва підходи дуже дорогі і тому для багатьох не підходили. Саме за цих умов і з'явилися системи NewSQL.

Сьогодні ландшафт в області NewSQL продовжує змінюватися, а стандартів поки що немає [9]. Цій темі присвячено чимало наукових статей, але на жаль, нерідко вони швидко старіють. Найчастіше актуальну інформацію можна знайти на сайтах відповідних систем та в блогах. Тому в цій статті намагатимемося усунути певною мірою цю прогалину, сконцентрувавши увагу на основних принципах, архітектурі та особливостях СКБД даного класу, а не на деталях реалізації, які з часом змінюються. При цьому для кращого розуміння викладеного матеріалу в роботі наводяться деякі теоретичні відомості, пов'язані з аспектами тематики, що розглядається, а також уточнюються деякі синонімічні терміни, в назві та перекладах яких у відомих релевантних джерелах присутня певна неоднозначність. Наприкінці наводяться деякі важливі узагальнені характеристики NewSQL СКБД для масштабованих рішень, що відрізняють їх від традиційних реляційних (RDBMS – Relational Database Management System) та NoSQL систем керування базами даних.

1. Основні категорії NewSQL баз даних

NewSQL – це клас сучасних систем керування реляційними базами даних, які прагнуть забезпечити високу продуктивність та масштабованість систем NoSQL, зберігаючи при цьому гарантії ACID (Atomicity, Consistency, Isolation, Durability) традиційної системи баз даних. Термін NewSQL був запропонований у 2011 р. аналітиком 451 Group (дослідницька компанія у сфері високих технологій, що базується в Нью-Йорку) Метью Аслетом [10]: «NewSQL – це скорочена назва різних нових постачальників масштабованих / високопродуктивних баз даних SQL. Раніше ми називали ці продукти «ScalableSQL», щоб відрізнити їх від існуючих продуктів реляційних баз даних. Оскільки це передбачає горизонтальну масштабованість, яка не обов'язково є функцією всіх продуктів, ми прийняли термін NewSQL у новому звіті».

Потреба в таких системах виникла в першу чергу у компаній (для їх корпоративних систем), які обробляють важливі дані (наприклад, фінансові системи, системи обробки замовлень і т. д.), яким потрібні рішення, що масштабуються, в той час як рішення NoSQL не могли забезпечити транзакційні механізми та не відповідали вимогам узгодженості даних. Серед застосувань систем NewSQL – розпізнавання шахрайства в реальному часі, електронна реклама, призначення розцінок та перехресні продажі в реальному часі, підтримка прийняття геопросторових рішень, proximity маркетинг, моніторинг ризиків, ціноутворення в реальному часі, Інтернет речей (IoT, internet of things) тощо [2, 11].

Системи баз даних NewSQL мають різні особливості та архітектури. Однак можна виділити їх такі спільні риси [11]: підтримка реляційної моделі даних та стандартного SQL; підтримка ACID транзакцій; масштабованість за рахунок поділу даних у кластерах без спільного використання ресурсів (shared nothing); та висока доступність за рахунок реплікації даних.

Розглянемо сучасні СКБД NewSQL з урахуванням їхнього групування на основі суттєвих аспектів їх реалізації. Для чого виділимо кілька категорій, які, на думку деяких дослідників, сьогодні найкраще являють собою системи NewSQL [6, 12]:

1. Системи, що створені з нуля з використанням нової архітектури.

2. Програмне забезпечення проміжного рівня (middleware), що дозволяє забезпечити можливість прозорого функціонування із сегментованою/фрагментованою на кількох вузлах (після так званого прозорого поділу – transparent sharding) бази даних.

3. Пропозиції «база даних як послуга» від постачальників хмарних обчислень, які також ґрунтуються на нових архітектурах.

1.1. Системи з використанням нової архітектури

У цю категорію включені системи NewSQL, розроблені на основі нової кодової бази, не спираючись на архітектуру застарілих систем, тобто створені з нуля. Усі СКБД цієї категорії засновані на розподілених (distributed) архітектурах, які працюють із ресурсами без поділу (спільного використання) та містять компоненти для підтримки управління паралельною роботою на кількох вузлах, відмовостійкості (за рахунок реплікації), керування потоками та

розподіленою обробкою запитів. Архітектура без поділу ресурсів (shared-nothing architecture) – це архітектура розподілених обчислень, в якій кожен запит на оновлення задовольняється одним вузлом у комп'ютерному кластері, що складається з кількох вузлів, які не використовують спільні ресурси (процесор, пам'ять, пристрій зберігання). Вперше цей термін був запроваджений Майклом Стоунбрейкером у роботі [13] при перерахуванні найпоширеніших архітектур для багатопроцесорних систем з високою швидкістю транзакцій. Перевага використання таких СКБД, створених для розподіленої обробки, полягає в тому, що всі частини системи можна оптимізувати для багатовузлових середовищ (multi-node environments). З цією метою, зокрема, використовуються оптимізатор запитів та протокол зв'язку між вузлами. Наприклад, більшість СКБД NewSQL можуть відправляти внутрішньозапитні дані безпосередньо між вузлами, а не відправляти в центральне розташування, як у деяких системах ПЗ проміжного рівня.

Практично кожна з СКБД у цій категорії керує своїм власним первинним сховищем, що знаходиться або в пам'яті, або на диску. Це означає, що СКБД відповідає за розподіл бази даних за своїми ресурсами за допомогою спеціального механізму замість того, щоб покладатися на готову розподілену файлову систему (наприклад, HDFS – Hadoop Distributed File System) або структуру зберігання (наприклад, Apache Ignite). Це важливо, оскільки цей механізм дозволяє СКБД відправляти запит до даних, а не переносити дані в запит, що призводить до значно меншого мережного трафіку, оскільки передача запитів зазвичай вимагає менше мережного трафіку, ніж передача даних (а це не тільки кортежі, але також індекси та матеріалізовані уявлення) для обчислення [6, 14].

Управління власним сховищем також дозволяє СКБД використовувати складніші схеми реплікації, ніж це можливо з блоковою схемою реплікації, яка використовується в HDFS. Загалом це дозволяє цим СКБД досягати більш високої продуктивності, ніж у інших систем, які накладаються поверх інших існуючих технологій (наприклад, так звані системи «SQL на Hadoop», такі як Trafodion та Splice Machine, які забезпечують транзакції поверх Hbase). Але є й недоліки використання СКБД з урахуванням нової архітектури. Перш за все, багато організацій побоюються впроваджувати надто нові та неперевірені технології. При цьому слід також пам'ятати, що кількість людей, які мають досвід роботи із системою, набагато менша, ніж у постачальників популярніших СКБД. Також слід мати на увазі, що організація може втратити доступ до існуючих інструментів адміністрування та звітності. Деякі СКБД, такі як MariaDB Xpand (раніше Clustrix) і SingleStore (раніше MemSQL), щоб уникнути цієї проблеми підтримують сумісність із дротовим протоколом (wire protocol) MySQL.

Приклади систем з використанням нової архітектури: *MariaDB Xpand* [15] (раніше *Clustrix*; компанія Clustrix була придбана MariaDB у 2018 р. [16] і Clustrix був перейменований на MariaDB Xpand [17 – 19]; ClustrixDB більше недоступний як окремий об'єкт, тепер він входить до складу MariaDB Enterprise Server, розширюючи його можливості за рахунок розподіленої обробки даних та транзакцій, перетворюючи його на розподілену базу даних SQL, здатну масштабуватися до мільйонів транзакцій за секунду з архітектурою без загального доступу), *CockroachDB* [20], *Google Cloud Spanner* [8, 21, 22], *HyPer* [23], *SingleStore* (раніше *MemSQL* – з жовтня 2020 р. SingleStore) [24], *NuoDB* (у грудні 2020 р. придбана компанією Dassault Systemes) [25], *SAP HANA* [26, 27], *H-Store* (фінальна версія H-Store була анонсована в 2016 р., з тих пір ніяких нових розробок не було) [28]. Якщо ви хочете використовувати більш сучасну СКБД, засновану на архітектурі H-Store, вам слід використовувати *Volt Active Data* (раніше *VoltDB*; перейменована в 2022 р. [29, 30], *LeanXcale* [31].

1.2. Проміжне ПЗ, що забезпечує прозоре функціонування сегментованої на кількох вузлах БД

Термін проміжне програмне забезпечення (middleware – менеджер ресурсів, що пропонує свої застосунки для ефективного обміну та розгортання цих ресурсів у мережі [32]), було введено на початку 1980-х років. Сьогодні завдяки проміжному програмному забезпеченню

програміст має можливість реалізувати децентралізоване рішення замість того, щоб взаємодіяти і аналізувати різні компоненти [33]. Пропоновані рішення, що функціонують як прозорий шар для одновузлових систем, дозволяють застосункам та користувачам представляти базу даних як єдину централізовану одиницю, незважаючи на те, що остання розбита на кілька частин/сегментів і розподілена між кількома вузлами. При цьому фрагментація /сегментування відрізняється від технологій об'єднання/федералізації (federation) баз даних 1990-х років, оскільки кожен вузол [6]: працює з однією і тією ж СКБД; має лише частину загальної бази даних; не призначений для доступу та оновлення самостійно окремими застосунками.

Використання проміжного ПЗ дозволяє розподіляти запити, координувати транзакції, а також керувати розміщенням, реплікацією та секціонуванням даних між вузлами. Найчастіше на кожному вузлі СКБД встановлюється так званий шар прокладки (shim), який взаємодіє з проміжним програмним забезпеченням. Цей компонент відповідає за виконання запитів від імені проміжного програмного забезпечення у своєму локальному екземплярі СКБД та повернення результатів. В цілому, все це забезпечує застосунку можливість роботи з базою даних, представляючи її як єдину логічну БД без необхідності модифікації базової СКБД. Основною перевагою використання ПЗ проміжного рівня, що забезпечує прозоре функціонування сегментованої на кількох вузлах БД, є те, що воно найчастіше є простою заміною застосунку, який вже використовує існуючу СУБД з одним вузлом. Розробникам не потрібно вносити будь-які зміни до свого застосунку, щоб використовувати нову сегментовану (sharded) базу даних. Для сумісності з конкретною СКБД проміжне програмне забезпечення має підтримувати відповідний протокол обміну. Хоча проміжне ПЗ дозволяє компаніям легко масштабувати свою БД на кілька вузлів, у ранніх системах доводилося використовувати традиційну СКБД (наприклад, MySQL, Postgres, Oracle) на кожному вузлі. Але ці СКБД, як правило, засновані на диск-орієнтованій (disk-oriented) архітектурі, і тому вони, наприклад, не можуть використовувати диспетчер/менеджер зберігання (storage manager) або схему управління паралелізмом (concurrency), оптимізовану для зберігання даних, орієнтовану на пам'ять (memory-oriented), як у деяких системах NewSQL, побудованих на нових архітектурах. Дослідження показали, що успадковані компоненти диск-орієнтованих архітектур є значною перешкодою, яка не дозволяє традиційним СКБД масштабуватись, щоб використовувати переваги більшої кількості ядер центрального процесора та великих обсягів пам'яті [6, 34]. Підхід, пов'язаний з використанням проміжного програмного забезпечення, також може призвести до надмірного планування та оптимізації запитів до фрагментів даних, розміщених на окремих вузлах, для складних запитів, а саме один раз на проміжному ПЗ і один раз на окремих вузлах СКБД. Хоча, з іншого боку, це дозволяє кожному вузлу застосовувати свої власні локальні оптимізації для кожного запиту.

Приклади систем: *MariaDB MaxScale* [35], *ScaleArc* [36].

1.3. База даних як послуга

З розвитком індустрії хмарних обчислень та розвитком послуг, що надаються хмарою, база даних також може надаватися як відповідна послуга, яка називається «база даних як послуга» (database-as-a-service, DBaaS). Завдяки такій послузі організаціям не потрібно підтримувати СКБД ні на власному обладнанні, ні на віртуальній машині (virtual machine, VM), розміщеній у хмарі. Постачальник DBaaS відповідає за підтримку фізичної конфігурації бази даних, включаючи налаштування системи, реплікацію та резервне копіювання. Клієнту надається URL-адреса для підключення до СКБД, а також засоби для керування системою. Споживачі служби хмарних обчислень (клієнти DBaaS) платять за відповідну послугу залежно від обсягу спожитих ресурсів згідно з встановленим тарифом. При цьому, оскільки запити до баз даних дуже різняться по тому, як вони використовують обчислювальні ресурси, постачальники DBaaS зазвичай не вимірюють виклики запитів так само, як вони вимірюють операції в блокових службах зберігання (наприклад, Amazon S3, Google Cloud Storage). Замість цього

клієнти підписуються на цінову категорію, в якій вказано максимальний поріг використання ресурсів (зокрема, розмір сховища, обчислювальну потужність, виділену пам'ять), який гарантує їм постачальник [6].

Як і в більшості категорій служб хмарних обчислень, найбільші компанії є основними гравцями в області DBaaS (головним чином завдяки можливості забезпечити економію за рахунок масштабу). Але при цьому слід зазначити, що майже всі DBaaS просто надають керований екземпляр традиційної СКБД (наприклад, MySQL, Microsoft SQL Server) з одним вузлом. До таких постачальників хмарних обчислень можна віднести Google Cloud SQL (база даних як послуга на базі MySQL, PostgreSQL і Microsoft SQL Server; це повністю керована служба бази даних, яка допомагає настроювати, підтримувати, керувати та адмініструвати реляційні бази даних на Google Cloud Platform), Microsoft Azure SQL Database (це хмарний сервіс (як частина Microsoft Azure), що надає можливість зберігання та обробки реляційних даних, а також генерації звітності) та деякі інші. Однак такі системи, на думку деяких авторів [6], недоцільно відносити до систем NewSQL, оскільки вони використовують ті ж базові дискові СКБД, що базуються на архітектурах 1970-х років, навіть незважаючи на те, що деякі постачальники, такі як Microsoft, модернізували свої СКБД, щоб забезпечити найкращу підтримку мультиорендних (multitenant) розгортань [37]. Натомість, як NewSQL, вони пропонують розглядати тільки ті продукти DBaaS, які базуються на новій архітектурі. Яскравим представником таких продуктів є Aurora від Amazon для MySQL RDS (Amazon Aurora – це система управління реляційними базами даних, розроблена для хмари та сумісна з MySQL та PostgreSQL; Aurora доступна як частина служби реляційних баз даних (Relational Database Service, RDS) Amazon).

Використовуючи продукти цієї категорії, слід не забувати про можливі ризики їх застосування у компанії. Наприклад, клієнти таких компаній, як GenieDB, Xeround, які надавали «базу даних як послугу», були змушені шукати нового постачальника послуги з відповідним перенесенням даних через те, що постачальник DBaaS припинив свою діяльність.

Прикладом бази даних як послуга, яка доступна в цій категорії NewSQL станом на 2023 р., є *Amazon Aurora* [38].

2. Особливості СКБД NewSQL

Коли організація вирішує використовувати СКБД NewSQL, для ухвалення правильного рішення необхідно враховувати багато характеристик. Розглянемо основні характерні особливості NewSQL СУБД, щоб зрозуміти, що є новим і заслуговує на увагу в цих системах і як наявні можливості використовувати в подальшому при реалізації конкретних систем обробки даних.

2.1. Механізм реплікації

Основною властивістю технології NewSQL, яка викликає інтерес, є можливість ефективного функціонування баз даних на великому кластері. Залежно від вибраної моделі розподілу можна створити сховище даних, що забезпечує більшу доступність і надає можливість обробляти більший обсяг даних за більш інтенсивного трафіку операцій читання або запису, уникаючи навантаження та гальмування мережі. Існують два способи розподілу даних [3, 9, 39]: реплікація (replication) та сегментування/фрагментація (sharding або fragmentation).

Реплікація має на увазі копіювання одних і тих же даних на декількох вузлах, тобто при реплікації дані копіюються між кількома серверами, так, що кожен біт даних можна знайти в різних місцях. Фрагментація/сегментування означає розміщення різних даних на різних вузлах (поділ БД на секції/розділи, які називаються фрагментами), тобто різні дані розподіляються по кількох серверах, щоб кожен сервер діяв як окреме джерело підмножини даних. При цьому слід враховувати існування двох фундаментальних проблем проєктування [9]: як здійснювати фрагментацію і як оптимально розподілити/розмістити ці фрагменти. Який із цих способів або їх комбінація буде більш ефективною в конкретному випадку, залежить від специфікації проєкту БД. Наприклад, чи слід розділити конкретну таблицю на кілька розділів

або реплікувати її на кожному вузлі. Визначення оптимальної конфігурації для довільного застосунку – нетривіальне завдання, особливо для складного корпоративного застосунку з багатьма залежностями.

Перш ніж продовжити подальший виклад матеріалу, доцільно звернути увагу на той факт, що сьогодні в існуючих численних релевантних джерелах має місце певна неоднозначність у назвах деяких термінів та їх перекладів, пов'язаних із тематикою, що розглядається. Наприклад, те, що в одних джерелах називається секцією (partition), в MongoDB, Elasticsearch і SolrCloud називається «шард» (shard), в HBase – «регіон» (region), у BigTable – «сегмент» (tablet), в Cassandra і Riak – «віртуальний вузол» (vnode), в Couchbase – «віртуальна ділянка» (vBucket). Термін секціонування (partitioning) – являє собою спосіб умисного розбиття великого набору даних на менші, пов'язується з терміном шардинг (sharding) [40] і т. д. Щодо термінології, пов'язаної з реплікацією, то тут теж усе достатньо переплетено. У різних джерелах можна зустріти такі синонімічні назви, як реплікація типу «master-slave» («головний-підлеглий»/«ведучий-ведений»), також відомою під назвою «leader-based replication» – «реплікація з провідним вузлом» або реплікація типу active-passive – «активний-пасивний» [40]). При розподілі за схемою «master-slave» відбувається реплікація даних по багатьох вузлах. Один вузол призначається головним (master або primary). Цей головний вузол є надійним джерелом даних і зазвичай відповідає за виконання всіх модифікацій цих даних. Інші вузли є підлеглими/веденими (*slaves*) або вторинними (secondaries). Процес реплікації синхронізує підлеглі вузли із головними [3]. При цьому типі реплікації запит спочатку обробляється на одному вузлі, а потім СКБД передає результуючий стан іншим реплікам. Існуючий тип реплікації з кількома головними вузлами (multi-leader, multi-master) називають також реплікацією типу «головний-головний» (master-master) або реплікацією типу «активний-активний» (active-active) [40]. За такої схеми кожен з головних вузлів одночасно є підлеглим для інших головних. При цьому типі реплікації кожен вузол репліки одночасно обробляє один і той же запит. Наприклад, коли транзакція виконує запит, СКБД виконує цей запит паралельно на усіх репліках. Однорангову реплікацію (peer-to-peer) називають симетричною (symmetric) чи оновлення будь-якої копії (update-everywhere) [39]. При одноранговій реплікації всі репліки мають однакову вагу, усі можуть виконувати операції запису, і втрата будь-якої з них не призводить до втрати доступу до сховища даних [3]. Тому в цій роботі, як правило, вказуватиметься декілька термінів-синонімів, пов'язаних із конкретним контекстом.

Реплікація баз даних – давній предмет вивчення, її принципи не сильно змінилися з 1970-х років, коли їх тільки почали вивчати, оскільки фундаментальні обмеження мереж залишилися тими самими [40].

Найкращий спосіб, за допомогою якого організація може забезпечити високу доступність та надійність даних, наприклад, для свого OLTP-застосунку, – це реплікувати свою базу даних. Усі сучасні СКБД, включаючи системи NewSQL, підтримують той чи інший механізм реплікації. При цьому DBaaS мають явну перевагу в цій галузі, оскільки вони приховують від своїх клієнтів усі найдрібніші деталі налаштування реплікації. Вони спрощують розгортання реплікованої СКБД, при цьому адміністратору не потрібно турбуватися про передачу журналів та перевірку синхронізації вузлів.

Коли справа доходить до реплікації бази даних, є два конструктивні рішення, що залежать від того, як СКБД забезпечує узгодженість даних між вузлами, а саме, яку модель узгодженості вона підтримує [6]: модель сильної узгодженості (strong consistency) або модель кінцевої узгодженості (eventual consistency) – іноді званою слабко узгодженою (weakly consistent) моделлю. У СКБД, що підтримує модель сильної узгодженості, записи транзакції мають бути підтверджені та встановлені на всіх репліках, перш ніж ця транзакція вважатиметься зафіксованою (тобто довговічною – durable). Перевага цього підходу полягає в тому, що репліки можуть обслуговувати запити лише для читання та при цьому залишатись узгодженими. Тобто, якщо застосунок отримує підтвердження того, що транзакцію зафіксовано, то будь-які зміни, зроблені цією транзакцією, видно будь-якій подальшій транзакції в майбу-

тньому, незалежно від того, до якого вузла СКБД вони звертаються. Це також означає, що при збої репліки немає втрачених оновлень, оскільки всі інші вузли синхронізуються. Але для підтримки цієї синхронізації потрібно, щоб СКБД використовувала протокол атомарної фіксації (наприклад, двофазну фіксацію – two-phase commit, 2PC [41]), щоб гарантувати, що всі репліки узгоджуються з результатом транзакції, що має додаткові витрати і може призвести до зупинки у разі збою вузла. Також необхідно враховувати, що два різні оновлення на головному вузлі можуть бути виконані безпосередньо одне за одним, залишаючи так зване вікно неузгодженості (inconsistency window) на кілька мілісекунд. Однак затримка в роботі мережі означає, що на підлеглих вузлах вікно неузгодженості залишиться відкритим набагато довше [3]. Ось чому, наприклад, системи NoSQL вибирають слабо узгоджену модель, в якій не всі репліки повинні підтверджувати зміну, перш ніж СУБД повідомить про успішний запис.

Відомі системи NewSQL підтримують сильно узгоджену реплікацію. Наприклад, Spanner і CockroachDB забезпечують схему реплікації, оптимізовану для сильно узгоджених реплік глобальної мережі [8, 42, 43]. У Spanner [8] це досягається за рахунок синхронізації, що забезпечується за допомогою комбінації GPS [44] та атомного годинника. Щоб мінімізувати похибку годинника Google встановлює GPS-приймач або атомний годинник у кожному центрі обробки та зберігання даних (data center), завдяки чому годинник синхронізується з точністю в межах 7 мілісекунд [40]. У CockroachDB це досягається завдяки синхронізації, що забезпечується за рахунок особливого використання часових міток/позначок часу поточного часу (wall time, real-world time, wall-clock time) вузла, максимального зміщення часу для кластера та періодичного порівняння зміщення часу вузлів кластера між собою [45].

У принципі, у тому, як ці системи забезпечують таку узгодженість, немає нічого нового. Основи реплікації кінцевого автомата для СКБД вивчалися ще 1970-х роках [46]. Комерційна система управління реляційними базами даних, що забезпечує відмовостійкість та масштабованість NonStop SQL, була однією з перших розподілених СКБД, створених у 1980-х роках, що використовують реплікацію сильної узгодженості для забезпечення відмовостійкості [47]. Сьогодні традиційні реляційні бази даних намагаються забезпечити сильну узгодженість, уникаючи всіх можливих неузгодженостей [3].

Більшість СКБД NewSQL реалізують реплікацію типу master-slave (active-passive), оскільки використовують так звану недетерміновану схему управління паралелізмом. Це означає, що вони не можуть надсилати запити до реплік у міру їх надходження на головний вузол, тому що вони можуть виконуватися в іншому порядку на репліках, і стан баз даних буде різнитися на кожній репліці. Це пов'язано з тим, що їхній порядок виконання залежить від кількох факторів, у тому числі від мережевих затримок, «зависань» кешу та неузгодженості годинників. З іншого боку, СКБД, які підтримують детерміновану схему управління паралелізмом, наприклад, Volt Active Data (H-Store/VoltDB), не виконують цих додаткових кроків координації. Це пов'язано з тим, що СКБД гарантує, що операції транзакцій виконуються в тому самому порядку на кожній репліці, і, таким чином, стан бази даних гарантовано буде однаковим [48].

Одним із важливих аспектів систем NewSQL є також можливість реплікації по глобальній мережі (Wide Area Network, WAN). Будь-яку СКБД NewSQL можна налаштувати для забезпечення синхронного оновлення даних по глобальній мережі, але це спричинить значне уповільнення нормальної роботи. Таким чином, замість цього СКБД NewSQL підтримують асинхронні методи реплікації (оновлення однієї репліки поширюється на інші через деякий час, а не в тій же транзакції).

Реплікація та фрагментація є ортогональними методами. Використовувати можна будь-який із них або обидва одночасно. Якщо ви використовуєте реплікацію master-slave (active-passive) та фрагментацію, це означає, що у вас є кілька головних вузлів, але кожна одиниця даних має лише один головний вузол. Залежно від конфігурації, ви можете призначити вузол головним для одних даних і підлеглим для інших або поєднати обов'язки головного та підле-

глого на одному вузлі. Наприклад, використання однорангової реплікації та фрагментації – поширена стратегія у базах даних NoSQL типу «широко-колонкове сховище» (wide column store / column families) [3].

2.2. Секціонування (розбиття або шардинг)

Подібно до того, як зростаючий бізнес повинен розширюватися, щоб задовольнити потреби своїх клієнтів, масштабуватися повинні й застосунки, що сприяють на рівні інформаційних технологій розвитку цього бізнесу. Забезпечення масштабованості бази даних є одним із пріоритетних завдань для розробників сучасних застосунків. Про масштабованість починають замислюватися, коли у сервера виникають труднощі зі збільшенням навантаження [49]. Рано чи пізно збільшене навантаження на сервер призведе до появи вузьких місць і, як наслідок, до зниження продуктивності. Вертикальне масштабування, як зазначалося раніше, є надто важким та витратним. Найбільш привабливим рішенням є горизонтальне масштабування – розміщення бази даних на кластері серверів.

Майже всі розподілені системи керування базами даних NewSQL масштабуються шляхом розбиття бази даних на непересічні підмножини, відомі як розділи (partitions) або шарди (shards), причому кожен розділ розташовується на різних серверах. Існує два основних типи фрагментації/секціонування [9, 39, 50, 51]:

– вертикальне розбиття (vertical partitioning) – передбачає розподіл стовпців таблиці на дві чи більше таблиць, пов'язаних первинним ключем вихідної таблиці; отримані стовпці поділяються між безліччю вузлів (рис. 1);

горизонтальне розбиття (horizontal partitioning) – передбачає поділ рядків таблиці між двома чи більше таблицями з однаковою структурою (рис. 2).

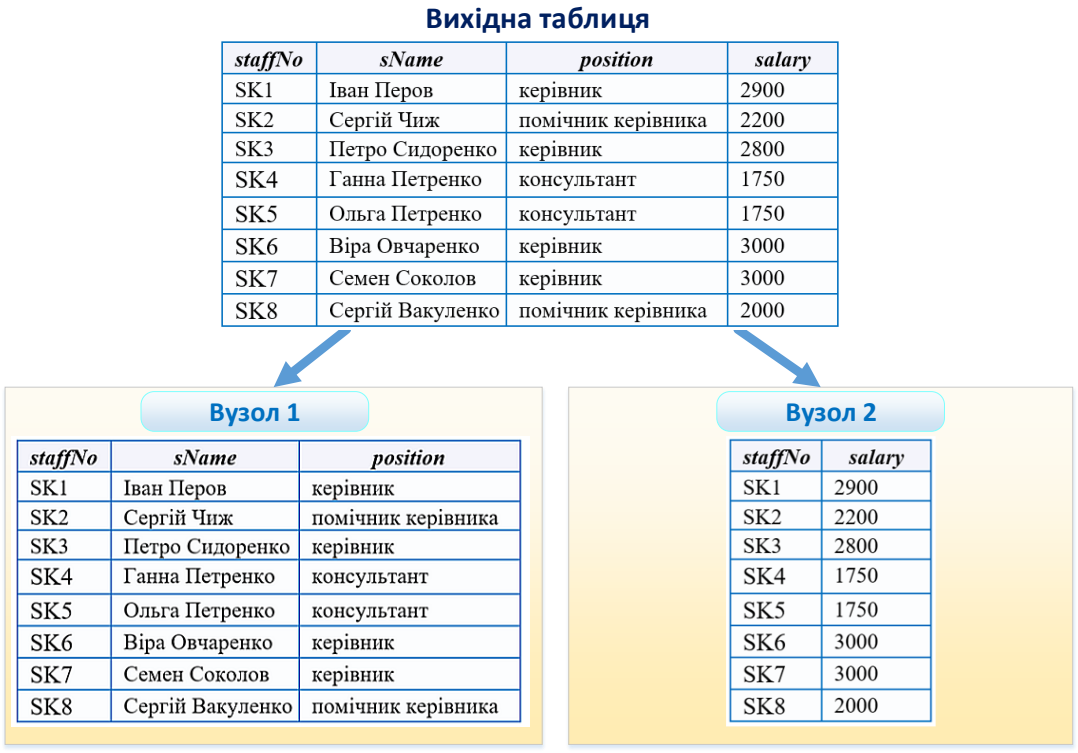

Рис. 1. Приклад вертикальної фрагментації

Основним способом, що знайшов більш широке застосування, масштабування баз даних для роботи на кількох вузлах є горизонтальна фрагментація [52]. Таблиці бази даних горизонтально поділяються на кілька фрагментів, межі яких ґрунтуються на значеннях одного (або кількох) стовпців таблиці (тобто атрибутів поділу).

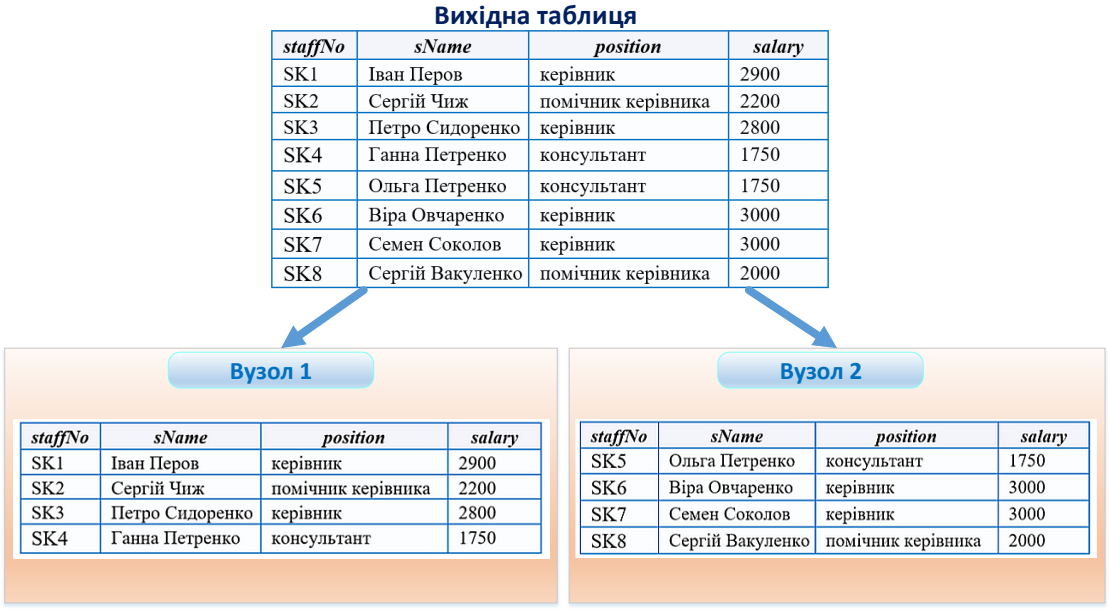

Рис. 2. Приклад горизонтальної фрагментації

СКБД надає кожному кортежу фрагмент (shard) на основі значень цих атрибутів, використовуючи один із методів. Пов'язані фрагменти кількох таблиць об'єднуються разом, щоб сформувати розділ, керований одним вузлом. Цей вузол відповідає за виконання будь-якого запиту, який потребує доступу до даних, що зберігаються в його розділі. Тільки системи DBaaS (зокрема Amazon Aurora) не підтримують цей тип розбиття [6].

Найбільш широко відомими та використовуваними підходами автоматичного горизонтального секціонування (розбиття) є [9, 52]:

– кругове розбиття (round-robin partitioning; стратегія, що забезпечує рівномірний розподіл даних; при n розділах/секціях, i-й кортеж у порядку вставки призначається розділу/секції (i mod n)),

– діапазонне розбиття (range partitioning; розділ/секція вибирається, виходячи з того, чи знаходиться ключ розбиття в межах певного діапазону; тобто розподіл кортежів здійснюється відповідно до набору предикатів),

– геш-розбиття (hash partitioning; застосовує геш-функцію до деякого атрибуту, в результаті застосування якої визначається номер розділу/секції).

Наприклад, геш-розбиття використовується в таких СКБД NewSQL, як Volt Active Data (VoltDB) та MariaDB Xpand (Clustrix). SAP HANA підтримує геш, діапазонне та кругове розбиття. LeanXcale підтримує діапазонне, геш розбиття та складове (composite) секціонування. Деякі СКБД NewSQL, такі як SingleStore (MemSQL), NuoDB і CockroachDB, мають власні моделі поділу, крім підтримки перерахованих вище методів секціонування.

Так, у СКБД NuoDB визначається один або кілька вузлів як менеджери/диспетчери зберігання (SM, storage managers), кожен з яких зберігає розділ бази даних. SM розбиває базу даних на блоки (звані мовою NuoDB «атомами» – atoms). Всі інші вузли в кластері позначаються як механізми транзакцій (TE, transaction engines – механізм транзакцій означає процес у базі даних, який надає клієнту застосунку доступ до бази даних [53]), які діють як кеш-пам'ять атомів. Для обробки запиту вузол TE витягує всі атоми, необхідні для цього запиту (або з відповідних SM, або з інших TE). TE встановлюють блокування запису на кортежі, а потім транслюють будь-які зміни до атомів іншим TE та SM. Щоб уникнути переміщення

атомів між вузлами туди та назад, NuoDB надає схеми балансування навантаження. NuoDB використовує ту ж схему розбиття, що й інші розподілені СКБД, але без необхідності попереднього секціонування бази даних або визначення зв'язків між таблицями.

SingleStore (MemSQL) також, як і NuoDB, використовує гетерогенну архітектуру (у якій не всі вузли однакові), що складається з вузлів-агрегаторів (aggregator nodes), призначених тільки для виконання, та вузлів-листів (leaf nodes), у яких зберігаються фактичні дані. Різниця між цими двома системами полягає в тому, як вони скорочують об'єм даних, що витягуються з вузлів зберігання (storage nodes) на вузли виконання (execution nodes). У NuoDB механізми транзакцій (TE) кешують атоми, щоб зменшити обсяг даних, які вони зчитують з диспетчерів зберігання (SM). Вузли-агрегатори SingleStore не кешують жодних даних, але листові вузли виконують частини запитів, щоб зменшити обсяг даних, що надсилаються на вузли-агрегатори. Це неможливо в NuoDB, тому що SM це тільки сховище даних. Ці дві системи можуть додавати додаткові ресурси виконання до кластеру СКБД (вузли TE NuoDB, вузли агрегатора SingleStore) без необхідності повторного розбиття бази даних.

На відміну від NuoDB та SingleStore, СКБД Volt Active Data використовує гомогенну архітектуру, де кожен вузол зберігає дані та виконує запити. І сьогодні ще належить з'ясувати, яка з архітектур гетерогенна або гомогенна є кращою з точки зору продуктивності або операційної складності [6].

Ще один новий аспект поділу в системах NewSQL полягає в тому, що деякі з них підтримують динамічну/живу міграцію (live migration). Це дозволяє СКБД переміщувати дані між фізичними ресурсами для повторного балансування та зменшувати кількість гарячих точок (hotspots) або збільшувати/зменшувати ємність СКБД без переривання обслуговування. Це схоже на повторне балансування в системах NoSQL, але складніше, оскільки СКБД NewSQL має підтримувати гарантії ACID для транзакцій під час міграції [14, 54]. Для цього у СКБД використовуються два підходи. Перший полягає в організації бази даних у вигляді безлічі крупнозернистих/гранулярних (coarse-grained) віртуальних/логічних розділів, поширених за фізичними вузлами [55].

Потім, коли СКБД необхідно виконати повторне балансування (re-balance – процес переміщення навантаження з одного вузла в кластері на інший; існують різні способи перебалансування [40]), вона переміщує ці віртуальні розділи між вузлами. Цей підхід використовується в MariaDB Xpand (Clustrix), а також у системах NoSQL, таких як Cassandra та DynamoDB. Інший підхід полягає в тому, щоб СКБД виконувала більш точне перебалансування шляхом перерозподілу окремих кортежів або кортежних груп за допомогою діапазонного розбиття. Цей підхід використовується в H-Store [54]. Рішення з іншою (власною) моделлю (підходом) поділу притаманне СКБД NewSQL Google Cloud Spanner. Сьогодні Spanner широко використовується як система керування базами даних OLTP для структурованих даних у Google. Spanner обслуговує десятки мільйонів запитів за секунду у всіх своїх базах даних, керуючи сотнями петабайт даних [22].

Таким чином, розміщуючи розділи на різних вузлах, що дозволяє масштабуватися до сотень або навіть тисяч вузлів [56], часто можна досягти майже лінійного прискорення, особливо для аналітичних запитів, коли кожен вузол може сканувати свої розділи паралельно [52]. Крім того, секціонування/фрагментація також може підвищити доступність, гарантуючи, що у разі збою одного розділу інші розділи зможуть відхилити деякі транзакції. В ідеалі СКБД також повинна мати можливість розподіляти виконання запиту на декілька розділів, а потім об'єднувати їх результати в один результат. Майже всі системи NewSQL (за винятком, напевно, ScaleArc), які підтримують вбудований поділ, надають цю функціональність.

Розподілена обробка транзакцій у розділених на секції (*partitioned*) базах даних – ідея не нова. Багато основ цих систем було взято з роботи авторів, які брали участь у кінці 1970-х у роботі над проєктом SDD-1 [57]. Крім того, на початку 1980-х років колективи авторів, які створили дві одновузлові СКБД – System R та INGRES, також створили розподілені версії своїх відповідних систем [58, 59]. Але через об'єктивні обставини ці СКБД так і не знайшли

широкого застосування. По-перше, на той час обчислювальні системи були дуже дорогими, тому більшість організацій не могли дозволити собі розгорнути свою базу даних на кластері машин. А по-друге, банально – не було потреби застосунків у високопродуктивній розподіленій СКБД. На той час очікувана пікова продуктивність СКБД зазвичай вимірювалася від десятків до сотень транзакцій на секунду. В даний час, обидва ці припущення вже без сумніву невірні. А створення великомасштабного застосунку з інтенсивним використанням даних сьогодні вже не є рідкістю.

Однак, незважаючи на певні досягнення в галузі розподіленої обробки транзакцій у сучасних розділених на секції базах даних, слід враховувати, що: *а*) час та ресурси, необхідні для створення та підтримки сегментованої архітектури БД, можуть переважити переваги її використання; *б*) на жаль, для робочих навантажень, що складаються з невеликих транзакцій, які стосуються кількох записів, жоден із наведених вище підходів горизонтального розбиття не є ідеальним. Якщо здійснюється доступ до більш ніж одного кортежу, то кругове і геш розбиття зазвичай вимагають доступу до кількох вузлів. Виконання розподілених транзакцій знижує продуктивність у порівнянні з виконанням транзакцій локально. Діапазонне розбиття може бути ефективнішим, але для цього потрібно ретельний вибір діапазонів, що може бути важко зробити вручну. Завдання секціонування/розбиття (partitioning problem) стає ще складнішим, коли транзакції стосуються кількох таблиць, які необхідно розділити за межами транзакцій. Наприклад, важко розділити дані для веб-сайтів соціальних мереж, де схеми часто характеризуються безліччю відносин «багато до багатьох» [52]. Крім того, підтримка цілісності даних може суттєво ускладнитися, оскільки функціонально залежні дані можуть виявитися фрагментованими та розміщуватись на різних вузлах. Тому сьогодні багато фахівців у світі проводять додаткові дослідження для вирішення цих проблем.

2.3. Пам'ять сховища

Із самого початку всі системи СКБД використовували архітектуру зберігання на основі дисків. Тобто в цих системах передбачалося, що основне місце зберігання бази даних знаходиться на запам'ятовуючому пристрої тривалого збереження з блоковою адресацією, такому як SSD (Solid-State Drive – твердотілий накопичувач) або HDD (Hard Disk Drive – жорсткий магнітний диск або накопичувач на магнітних дисках). І оскільки операції читання та запису на ці пристрої виконуються повільно порівняно зі швидкістю основної пам'яті, СКБД змушена використовувати основну/оперативну пам'ять (main memory) для кешування блоків, що зчитуються з диска, і для буферизації оновлень, отриманих в результаті транзакцій. Використання подібних довготривалих запам'ятовуючих пристроїв було викликано тим, що історично основна пам'ять була набагато дорожчою і мала обмежену ємність у порівнянні з дисками. Однак сьогодні сучасні технології досягли такого рівня розвитку, коли потужності та ціни такі, що можна зберігати всі бази даних OLTP (крім дуже великих) повністю у пам'яті (так звані in-memory database – (IMDB), або система баз даних в основній пам'яті – main memory database system (MMDB), або резидентна база даних – memory resident database). Перевага цього підходу полягає в тому, що він дозволяє проводити певну оптимізацію, оскільки СКБД більше не доводиться припускати, що транзакція може отримати доступ до даних у будь-який момент часу, і якщо дані не перебувають у пам'яті, СКБД доведеться перейти в стан очікування. Таким чином, ці системи можуть підвищити продуктивність, тому що багато компонентів, необхідних для обробки у відповідних ситуаціях, наприклад менеджер пула буферів або схеми управління паралелізмом з високим пріоритетом, просто стануть не потрібними.

Ідея зберігання бази даних повністю в оперативній пам'яті не нова [60, 61]. Так, IMS Fast Path [62], випущена в 1976 р., є однією з перших відомих систем, оптимізованих для даних, що знаходяться в пам'яті. В рамках проєкту MM-DBMS Університету Вісконсин-Медісон (University of Wisconsin-Madison) [63 – 65] були проведені основні дослідження, які заклали основу для багатьох аспектів СКБД з оперативною пам'яттю, включаючи індекси, обробку

запитів та алгоритми відновлення. У тому ж десятилітті було розроблено перші розподілені СКБД, засновані на архітектурі зберігання в основній пам'яті (проєкт PRISMA – великомасштабна дослідницька робота Philips Research Laboratory; однією з його цілей була розробка та реалізація розподіленої машини бази даних з оперативною пам'яттю [66]). Нова хвиля таких СКБД спостерігалася у 1990-х роках. У цей час з'явилися такі комерційні СКБД, як Altibase [67], Oracle TimesTen [68], які і сьогодні продовжують функціонувати та підтримуватись розробниками. Деякі NoSQL системи також підтримують можливість зберігання даних у пам'яті, наприклад, Redis (REmote DIctionary Server) – це мережеве сховище ключів та значень у пам'яті, яке можна використовувати як базу даних, кеш або брокер повідомлень. Починаючи з MongoDB Enterprise версії 3.2.6, MongoDB додала механізм зберігання пам'яті в механізми зберігання. Набори реплік MongoDB допускають гібридне розгортання бази даних у пам'яті та на диску.

Потенційною технічною перешкодою використання СКБД, заснованих на архітектурі зберігання в основній пам'яті, вважатиметься можливість втрати живлення. Незалежно від того, з якої причини відбувається втрата живлення, енергозалежна оперативна пам'ять втрачає всі дані. Однак з появою енергонезалежної пам'яті з довільним доступом або так званої технології енергонезалежної пам'яті (NVM, Non-Volatile Memory, Persistent Main Memory, Storage-Class Memory) бази даних в оперативній пам'яті можуть забезпечувати таку ж продуктивність і зберігати дані в разі втрати живлення. Крім того, застосування технології енергонезалежної пам'яті дозволяє використовувати однорівневу архітектуру зберігання даних без використання будь-яких зовнішніх запам'ятовуючих пристроїв, на відміну від традиційних транзакційних in-memory СКБД, змушених використовувати постійне зовнішнє сховище, щоб забезпечити важливу характеристику транзакції – довговічність.

Існує кілька СКБД NewSQL, заснованих на архітектурі зберігання в основній пам'яті, такі, наприклад, як HyPer, SingleStore, SAP HANA, Volt Active Data (H-Store/VoltDB). Ці СКБД за рахунок орієнтації на основну пам'ять працюють значно ефективніше при робочих навантаженнях OLTP систем. Робочі навантаження (workload – обсяг роботи, яку людина чи машина мають виконати за певний період часу [69]) OLTP характеризуються невеликими наборами даних, і великою кількістю простих запитів. Звичайною мірою робочих навантажень OLTP є кількість транзакцій за секунду [70].

Однією з особливостей систем NewSQL, побудованих на архітектурі зберігання в основній пам'яті, є можливість витіснення підмножини бази даних у постійне сховище, щоб зменшити пам'ять, яку вона займає. Це дозволяє СКБД підтримувати бази даних, розмір яких перевищує обсяг доступної пам'яті без необхідності зворотного перемикання на дискову архітектуру.

Загальний підхід, який отримав назву антикешування (anti-caching) [71, 40], полягає у використанні внутрішнього механізму відстеження всередині системи, щоб визначити, до яких кортежів більше немає доступу, а потім вибрати їх для виключення. Тобто за такого підходу за відсутності достатньої кількості доступної пам'яті «холодні» (що найдавніше використовувалися) дані з основної пам'яті переміщаються на диск безпечним для транзакцій способом і завантажуються у оперативну пам'ять при зверненні у майбутньому. Таким чином, «гарячі» дані знаходяться в основній пам'яті, а більш «холодні» – на диску в антикеш-частині системи. Антикешування дозволяє СКБД, побудованих на архітектурі зберігання в основній пам'яті, керувати базами даних, розмір яких перевищує обсяг колективної пам'яті на всіх вузлах [71]. Це нагадує те, що операційні системи роблять з віртуальною пам'яттю і файлами підкачування, але СКБД може керувати пам'яттю ефективніше, ніж операційна система, за рахунок маніпуляцій на рівні окремих записів, а не цілих сторінок пам'яті [40]. Подібний підхід з деякою модифікацією, що дозволяє більш ефективно здійснювати переміщення між основною пам'яттю та сховищем тривалого збереження, використовується у Volt Active Data (H-Store/VoltDB). Існує підтримка багаторівневого антикешування, що дозволяє визначити ієрархічну структуру сховища для перенесення кортежів із DRAM (dynamic random

access memory) у сховище тривалого збереження [28]. Також додано підтримку міграції (переміщення з основної пам'яті та повернення до основної пам'яті) даних на рівні кортежу при використанні енергонезалежної пам'яті. Замість використання так званих «tombstone» (віддалений запис у репліці розподіленого сховища даних) виключені кортежі переміщаються в окремий пул, що зберігається в NVM. Таким чином, можна безпосередньо звертатися до цих кортежів, не перериваючи і не перезапускаючи транзакцію.

Ймовірно, коли ширше використовуватимуться технології енергонезалежної пам'яті, знадобляться подальші зміни в конструкції підсистем зберігання [40, 72]. В даний час це ще відносно нова область досліджень, але вона заслуговує на пильну увагу. У цьому контексті можна відзначити SAP HANA – першу велику СКБД, яка активно використовує NVM [73, 74], а саме можливостей енергонезалежної пам'яті з більш високою щільністю (Intel Optane PMem), яка дозволяє зберігати дані як твердотілі накопичувачі та накопичувачі на жорстких магнітних дисках і забезпечує швидкість, аналогічну до основної пам'яті. Pmem (Persistent Memory) забезпечує унікальне поєднання доступної великої ємності та підтримки збереження даних. Завдяки інноваційній технології, що пропонує різні режими роботи, вона адаптується до потреб залежно від робочих навантажень [75].

SingleStore (MemSQL) використовує інший підхід до баз даних, обсяг яких перевищує обсяг пам'яті. У ній адміністратор може вручну вказати СКБД зберігати таблицю у форматі стовпців (columnar format). Ця СКБД не підтримує жодних метаданих відстеження в пам'яті для цих резидентних на диску кортежів. Вона організує ці дані у сховищі з журнальною структурою (log-structured storage), щоб зменшити накладні витрати на оновлення, які традиційно виконуються повільно у сховищах даних OLAP (online analytical processing).

2.4. Управління паралельною обробкою

Одночасне виконання транзакцій у розрахованих на багато користувачів базах даних може викликати певні проблеми, пов'язані із забезпеченням цілісності (integrity) та узгодженості (consistency) даних, наприклад, такі як: втрачене оновлення, «брудне» читання, неповторюване читання, фантомне читання. Тому керування паралельною обробкою (паралелізмом – concurrency control) є важливою концепцією, яку мають підтримувати бази даних, у тому числі NewSQL. Мета керування паралельною обробкою – запобігання непередбаченому впливу дій одного користувача на дії іншого. Іншими словами, щоб з одного боку в умовах паралельної обробки користувач отримав той же результат, як і у випадку, якби він був єдиним користувачем, а з іншого – щоб дії різних користувачів, якби й впливали один на одного, то тільки прогнозованим чином.

На цей час розроблено різні підходи керування паралельною обробкою, зокрема [9]:

– схеми двофазного блокування (2PL, two-phase locking). При цій стратегії транзакціям дозволяється накладати блокування в міру необхідності, але як тільки перше блокування знімається, транзакція вже не може накласти ніяких інших блокувань;

– упорядкування за позначками часу/часовими мітками (timestamp ordering – TO). СКБД передбачає, що транзакції виконуються в порядку їх позначок часу, при цьому транзакції не будуть виконувати операції, що чергуються, які порушать порядок, що серіалізується. Цей протокол вимагає, щоб всі розподілені вузли мали точно синхронізовані годинники;

– керування паралельним доступом через багатоверсійність/багатоверсійне керування конкурентністю (MVCC, multi-version concurrency control). MVCC – це метод керування паралелізмом, який зазвичай використовується СКБД для забезпечення одночасного доступу до бази даних та мовами програмування для реалізації транзакційної пам'яті. Що стосується конкретно забезпечення паралельного доступу до баз даних, суть цього методу полягає в наданні кожному користувачеві «знімку» бази. При цьому даний «знімок» має таку властивість, що зміни, що вносяться користувачем, невидимі іншим користувачам до моменту фіксації транзакції. Цей спосіб керування дозволяє домогтися того, що транзакції, що пишуть, не блокують ті транзакції, що читають, і транзакції, що читають, не блокують ті, що пишуть.

Для цього MVCC використовує позначки часу та інкрементуючі ідентифікатори (ID) транзакцій для досягнення узгодженості транзакцій. MVCC гарантує, що транзакції ніколи не доведеться чекати на читання об'єкта бази даних, підтримуючи кілька версій об'єкта. Кожна версія об'єкта (кортежу) має позначку часу для читання та позначку часу для запису, що дозволяє конкретній транзакції прочитати останню версію об'єкта, яка передує позначці часу читання. Проблема MVCC – наявність кількох версій елементів даних із різними значеннями. Щоб заощадити місце, старі версії даних повинні періодично видалятися. Але це можна робити тільки коли розподілена СКБД впевнена, що більше не з'явиться транзакція, якій потрібен доступ до видалених версій [9].

Однак, на жаль, сьогодні жоден метод чи механізм керування паралельною обробкою не є ідеальним для всіх випадків [76]. Усі вони передбачають певний компроміс. Наприклад, застосунок може дуже жорстко керувати паралельною обробкою, заблокувавши всю базу даних, але жодні інші програми не зможуть нічого робити під час її виконання. Такий захист надійний, але дорогий. Разом з тим існують заходи, які складніше запрограмувати або реалізувати, але які забезпечують більшу пропускну здатність. Доступні й інші заходи, які максимізують пропускну здатність, але мають низький рівень керування паралельною обробкою. Тому при розробці розрахованих на багато користувачів баз даних доводиться робити вибір серед цих заходів, йдучи на певні компроміси.

І якщо перші розподілені СКБД 1970-80-х років використовували схеми двофазного блокування, то майже всі системи NewSQL, побудовані на нових архітектурах, уникають 2PL через складність роботи із взаємоблокуванням. Замість цього поточна тенденція полягає у використанні варіантів керування паралельною обробкою з упорядкуванням за позначками часу. Найбільш широко використовуваний протокол у системах NewSQL – це децентралізоване керування паралельним доступом у вигляді багатоверсійності (MVCC). Цей механізм використовується практично у всіх системах NewSQL, заснованих на нових архітектурах, зокрема SingleStore, HyPer, SAP HANA, CockroachDB, MariaDB Xpand, LeanXcale. При цьому, незважаючи на інженерні оптимізації та налаштування, які ці системи використовують у своїх реалізаціях MVCC для підвищення продуктивності, основні концепції схеми не є новими. Подібні рішення використовувалися, наприклад, у СКБД VAX Rdb/ELN та InterBase на початку 1980-х років [6].

Інші системи NewSQL для керування паралельною обробкою використовують комбінацію 2PL та MVCC. При такому підході транзакції, як і раніше, повинні накладати блокування за схемою 2PL для модифікації бази даних. Коли транзакція змінює запис, СКБД створює нову версію цього запису так само, як у випадку з MVCC. Ця схема дозволяє запитам лише для читання уникнути необхідності накладання блокувань і, отже, не блокувати транзакції записи. Найбільш відомою реалізацією цього підходу є механізм зберігання (storage engine) InnoDB для СКБД MySQL. Також ця схема використовується в Spanner, NuoDB та MariaDB Xpand (Clustrix).

NuoDB покращує вихідний MVCC, використовуючи gossip/epidemic протокол для широкомовної передачі інформації про версії між вузлами. CockroachDB використовує для керування паралельною обробкою та забезпечення рівня ізоляції «серіалізований» (serializable) різновид MVCC [77]. Комерційна СКБД NewSQL Volt Active Data використовує керування паралельною обробкою на основі методу позначок часу, але замість чергування транзакцій, як і в MVCC, вона планує виконання транзакцій по одній за раз у кожному розділі. Вона також використовує гібридну архітектуру, в якій транзакції з одним розділом плануються децентралізованим чином, а транзакції з кількома розділами плануються за допомогою централізованого координатора (коли запит на транзакцію надходить на вузол, координатор надає запиту на унікальний ідентифікатор на основі позначки часу його надходження). Volt Active Data (H-Store/VoltDB) упорядковує транзакції на основі логічних позначок часу, а потім планує їх виконання у розділі, коли настає їхня черга. Коли транзакція виконується у розділі, вона має ексклюзивний доступ до всіх даних у цьому розділі, і, таким чином, системі

непотрібно встановлювати детальні блокування (fine-grained locks) та засувки (latches) для своїх структур даних. Це дозволяє транзакціям, яким потрібний доступ тільки до одного розділу, ефективно виконуватись, оскільки інші транзакції не конкурують одна з одною.

Недоліком підходу керування паралелізмом на основі розділів є те, що він не працює належним чином, якщо транзакції охоплюють кілька розділів, оскільки затримки мережного обміну даними змушують вузли не діяти в очікуванні повідомлень [6]. Слід зазначити, що керування паралельною обробкою на основі розділів також не є новою ідеєю. Вперше цей підхід було запропоновано у роботі [78].

Все проміжне програмне забезпечення та служби DBaaS успадковують схему керування паралелізмом архітектури СКБД, що лежить в їх основі. А оскільки більшість із них використовують MySQL, для керування паралельною обробкою вони використовують комбінацію 2PL та MVCC. В цілому, слід зазначити, що в основних схемах керування паралельною обробкою в системах NewSQL немає нічого принципово нового. Але це не зменшує їхньої значущості, оскільки реалізації в них відомих механізмів дозволяють ефективно функціонувати системі в контексті сучасного обладнання та розподілених операційних середовищ.

2.5. Вторинні індекси

Строго кажучи, індекси в цілому не є обов'язковим компонентом СКБД, але вони можуть істотно підвищити її продуктивність. Використання вторинного індексу – це спосіб ефективного доступу до записів у базі даних за допомогою деяких атрибутів таблиці, відмінних від атрибутів первинного ключа. Їх просто підтримувати у нерозділених на секції (non-partitioned) СКБД, оскільки вся база даних розташована на одиночному вузлі. Проблема з вторинними індексами виникає у розподіленій СКБД. Вона полягає в тому, що вторинні індекси не завжди можуть бути розділені так само, як і решта бази даних. Існуючі сьогодні конструктивні рішення для підтримки вторинних індексів у розподіленій СКБД залежать від того:

- де система їх зберігатиме;
- як вона підтримуватиме їх у контексті транзакцій.

У системі з централізованим координатором, як і в ПЗ проміжного рівня, що забезпечує прозоре функціонування з сегментованою на кількох вузлах БД, вторинні індекси можуть перебувати як на вузлі координатора, так і на вузлах сегментів. Перевага цього підходу полягає в тому, що у всій системі існує лише одна версія індексу, і тому його підтримувати легше. Усі системи NewSQL, засновані на нових архітектурах, децентралізовані та використовують секційовані вторинні індекси. Це означає, що кожний вузол зберігає частину індексу, а не повну копію. Компроміс між секційованими та реплікованими індексами полягає в тому, що з запитами, що мали місце, можливо, знадобиться охопити декілька вузлів, щоб знайти те, що вони шукають, але, якщо транзакція оновлює індекс, їй потрібно буде його змінити тільки на одному вузлі. У реплікованому індексі ролі міняються місцями. Пошуковий запит може бути задоволений лише одним вузлом у кластері, але щоразу, коли транзакція змінює атрибути, вказані у базовій таблиці вторинного індексу (ключ чи значення), СКБД має виконувати розподілену транзакцію, яка оновлює всі копії індексу.

Прикладом децентралізованого вторинного індексу, в якому поєднуються ці концепції, є MariaDB Xpand. Спочатку СКБД підтримує реплікований крупнозернистий (coarse-grained), заснований на діапазоні індекс на кожному вузлі, який відображає значення в розділи/частини (partitions). Це відображення дозволяє СКБД надсилати запити до відповідного вузла, використовуючи атрибут, який не є атрибутом розбиття таблиці. Потім ці запити будуть звертатися до другого секційованого індексу на цьому вузлі, який відображає точні значення в кортежі. Такий дворівневий підхід знижує об'єм координації, необхідної для синхронізації реплікованого індексу в кластері, оскільки він відображає лише діапазони, а не окремі значення. Додавання вторинного індексу до стовпців таблиць бази даних Google Cloud Spanner підвищує ефективність пошуку даних у цих стовпцях, у тому числі за рахунок ска-

нування індексу, а не повного сканування таблиці. Spanner зберігає такі дані у кожному вторинному індексі [79]: все ключові стовпці з базової таблиці; всі стовпці, включені до індексу; усі стовпці, зазначені в необов'язковому реченні STORING (діалект SQL для Google) або у реченні INCLUDE (діалект для PostgreSQL) визначення індексу. З часом Spanner аналізує таблиці, щоб упевнитися, що вторинні індекси використовуються для відповідних запитів. LeanXcale підтримує ефективні вторинні індекси, які розподіляються та зв'язуються з первинними даними. Запити до вторинних індексів можуть витягувати первинні дані локально і надавати повний результат з мінімальною кількістю циклів обробки. У SAP HANA підтримується довільна кількість унікальних та неунікальних вторинних індексів. Усередині вторинні індекси перетворюються на два різні варіанти, залежно від кількості задіяних стовпців [80]:

– індекси для окремих стовпців. Під час створення індексу для окремого стовпця сховище стовпців створює інвертований список (інвертований індекс – inverted index), який відображає ідентифікатори значень словника у відповідні записи у векторі індексу. Усередині створюються дві структури індексу: одна для дельта-таблиці та одна для основної таблиці (кожна таблиця сховища стовпців складається з двох окремих частин: основної таблиці та дельта-таблиці; у той час як основна таблиця призначена тільки для читання, сильно стиснута та оптимізована для читання, дельта-таблиця відповідає за відображення змін, виконаних операціями INSERT, UPDATE, DELETE). Коли цей індекс створюється для сховищ рядків, створюється лише один окремий індекс B+ дерева;

– індекси за кількома стовпцями (конкатеновані / складові індекси – concatenated indexes). Індекс з кількома стовпцями може бути корисним, якщо часто запитується певна комбінація атрибутів, або для прискорення обробки з'єднання, коли задіяно кілька атрибутів. Слід звернути увагу на те, що при створенні складеного / конкатенованого індексу окремі індекси для складових атрибутів не створюються (на відміну від первинного ключа, де також створюються індивідуальні індекси для кожного із складових атрибутів). Сховище стовпців підтримує інвертований індекс значень, інвертований геш-індекс та інвертований індивідуальний індекс для багатостолбцових індексів (multi-column indexes). При створенні складеного / конкатенованого індексу для сховища рядків створюється лише один окремий індекс B+ дерева.

Найбільш поширений спосіб створення вторинних індексів розробниками при використанні СКБД NewSQL, яка їх не підтримує – це застосування індексу з використанням розподіленого кеша в пам'яті, такого як Memcached [81]. Але використання зовнішньої системи вимагає, щоб застосунок підтримував кеш, оскільки СКБД не будуть автоматично скасовувати зовнішній кеш [6].

2.6. Механізм відновлення після збою

Важливою характеристикою СКБД NewSQL, пов'язаної із забезпеченням відмовостійкості (fault tolerance), є її механізм відновлення після збоїв (crash recovery). При цьому на відміну від традиційних СКБД, де основним завданням відмовостійкості є гарантія відсутності втрати оновлень, СКБД NewSQL також повинні мінімізувати час простою [6]. Так як передбачається, що сучасні веб-застосунки постійно будуть перебувати в мережі, а перебої в роботі сайту обходяться дорого.

Традиційний підхід до відновлення в системі з одним вузлом без реплік полягає в тому, що коли СКБД повертається в оперативний (online) режим після збою, вона завантажує останню контрольну точку (точку синхронізації між базою даних та журналом транзакцій), а потім відтворює свій журнал із записом наперед (WAL, write-ahead log) для повернення стану бази даних у той стан, у якому вона була на момент збою. Класичний метод цього підходу, відомий як ARIES (Algorithm for Recovery and Isolation Exploiting Semantics) [82], було винайдено дослідниками IBM у 1990-х роках. Усі основні СКБД реалізують той чи інший варіант ARIES, який підтримує часткові відкочування транзакцій, блокування з високим ступенем деталізації (наприклад, запис) та відновлення з використанням ведення журналу із

записом наперед. Однак у розподіленій СКБД з репліками традиційний підхід з одним вузлом не застосовується безпосередньо. Це пов'язано з тим, що під час збою головного вузла система активізує один із підлеглих вузлів в якості нового головного. Коли попередній головний вузол повертається до online режиму, він може просто завантажити свою останню контрольну точку і повторно запустити свій WAL, оскільки СКБД продовжує обробляти транзакції і, отже, стан бази даних просунувся вперед. Відновлювальному вузлу необхідно отримати оновлення від нового головного вузла (і, можливо, інших реплік – вузлів, на якому зберігається копія бази даних), які він пропустив, коли був у неробочому стані. Є два можливі способи зробити це. По-перше, відновлюючий вузол може завантажити свою останню контрольну точку та WAL зі свого локального сховища, а потім запросити зміни, які він пропустив із записів журналів інших вузлів. Поки вузол може обробляти журнал швидше, ніж до нього додаються нові оновлення, вузол врешті-решт прийде до того ж стану, що й інші реплік вузли. Це можливо, якщо СКБД використовує журнал із записом наперед, оскільки час застосування оновлень журналу безпосередньо до кортежів набагато менше, ніж час, необхідний для виконання вихідного оператора SQL. Інший варіант, що дозволяє скоротити час, необхідний для відновлення, передбачає відмову вузла, що відновлюється, від своєї контрольної точки на користь використання нової точки, з якої вузол буде відновлюватися. Тобто вузол, що відновлюється, може прийняти поточний стан нового головного вузла і продовжити роботу в якості звичайного вузла. Додаткова перевага цього підходу полягає в тому, що цей механізм можна використовувати і в СКБД для додавання нового вузла репліки.

LeanXcale надає механізм гарячого резервного копіювання (hot backup), що дозволяє виконувати повне резервне копіювання (full backup) розподіленої бази даних, гарантуючи послідовне відновлення на певний момент часу.

Ведення журналу в системах, що базуються на архітектурі зберігання в основній пам'яті, оптимізована для забезпечення високої пропускної здатності та низької затримки [61]. Оскільки введення-виведення журналу є основним вузьким місцем, ці системи намагаються максимально зменшити обсяг журналу навіть більше, ніж системи на основі дисків. Volt Active Data (H-Store/VoltDB) використовує спрощений варіант ведення журналу, званого журналом команд (command log) [48, 83], який реєструє лише запит на запуск транзакції (збереженої процедури). Тобто виклик кожної збереженої процедури являє собою транзакцію. Одна збережена процедура може включати безліч окремих операторів SQL, і кожен оператор SQL може змінювати сотні або тисячі рядків таблиці. Запис журналу для цієї схеми складається виключно з імені збереженої процедури, її вхідних параметрів та ідентифікатора транзакції. Процес протоколювання команд (ведення журналу команд) представлено на рис. 3. Частота (frequency), з якої транзакції записуються до журналу команд, налаштовується. Регулюючи частоту та тип ведення журналу (синхронний або асинхронний), ви можете збалансувати потреби вашого застосунку у продуктивності з бажаним рівнем надійності [83].

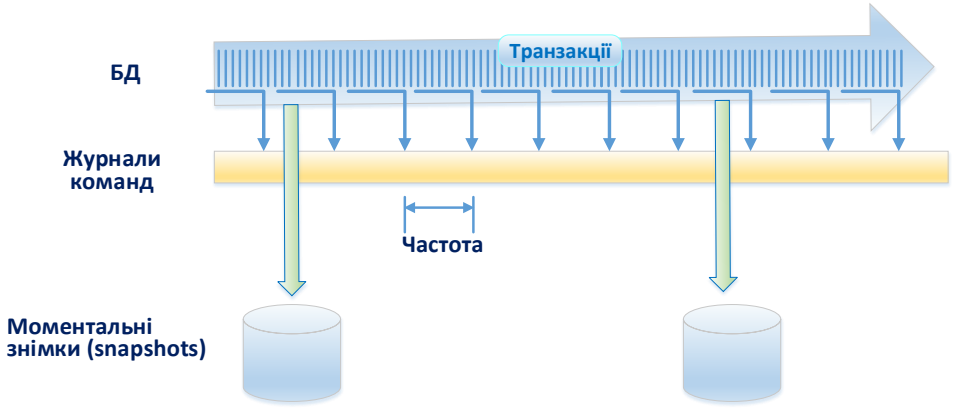

Рис. 3. Процес протоколювання команд (ведення журналу команд)

Щоб ще більше звести до мінімуму трафік журналу, деякі системи взагалі уникають реєстрації даних індексу. Реєструються лише оновлення базових даних. Після збою індекси будуються із нуля після відновлення базових записів. Крім того, СКБД, засновані на архітектурі зберігання в основній пам'яті, намагаються максимально розпаралелити введення-виведення журналів. На додаток, оскільки контрольні точки у цих системах зазвичай більше, ніж їх аналоги в дискових системах (це пов'язано з тим, що системи баз даних в основній пам'яті зазвичай записують усі (або більшість) рядків у сховище), були розроблені полегшені методи створення контрольних точок в основній пам'яті, наприклад, CALC (Checkpointing Asynchronously using Logical Consistency) [84], який переходить в режим копіювання при записі, записуючи тільки оновлення, що відбулися з моменту попередньої контрольної точки. Volt Active Data (H-Store/VoltDB) також використовує аналогічну схему копіювання під час запису для створення асинхронних моментальних знімків даних на кожному вузлі. Резервні копії та контрольні точки (РККТ) в in-memory базах даних, необхідні для забезпечення відмовостійкості БД, повинні зберігатися на постійних пристроях зберігання, таких як SSD або жорсткі диски. База даних повинна гарантувати актуальність резервних копій та мінімальний час відновлення [14].

У більшості СКБД, заснованих на архітектурі зберігання в основній пам'яті, дані відновлюються шляхом завантаження останньої дійсної (valid) контрольної точки з подальшим відтворенням кінця журналу, уникаючи повного скасування [61]. Це відноситься і до баз даних, встановлених на платформі СКБД Volt Active Data (H-Store/VoltDB), дані яких відновлюються шляхом завантаження останнього повного узгодженого моментального знімка (snapshot), після чого відтворюються (replay) транзакції зі свого журналу команд (починаючи з моменту останнього моментального знімка), щоб привести базу даних до узгодженого стану (рис. 4).

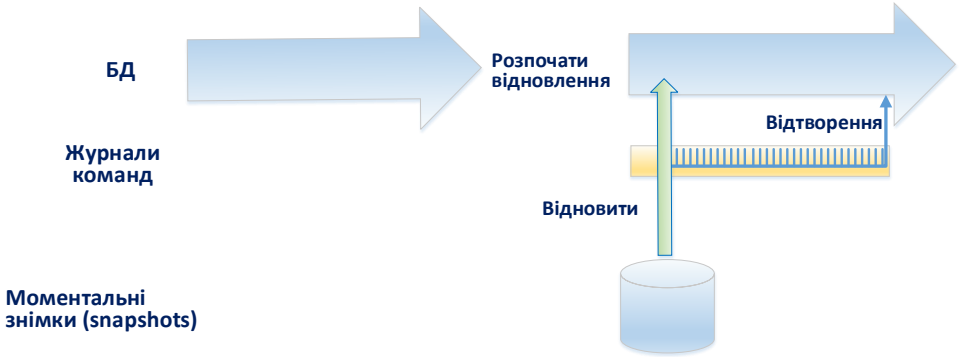

Рис. 4. Процес відновлення БД

Проміжне програмне забезпечення та системи DBaaS спираються на вбудовані механізми базових одновузлових СКБД, але додають додаткову інфраструктуру для вибору головного вузла та інші можливості керування. Системи NewSQL, засновані на новій архітектурі, використовують комбінацію готових компонентів (наприклад, ZooKeeper, Raft) та власних реалізацій існуючих алгоритмів (наприклад, Paxos) [61]. Все це стандартні процедури та технології, доступні у комерційних розподілених системах з 1990-х років.

2.7. Забезпечення безпеки

Безпека баз даних важлива для будь-якої організації користувача, які використовують великі набори даних як значущий актив. При цьому слід мати на увазі, що поняття безпеки відносяться не лише до даних, що зберігаються у базі даних. Порушення безпеки можуть торкнутися інших частин системи, що, своєю чергою, може вплинути на базу даних. Тобто безпека БД залежить від захищеності використовуваного обладнання, програмного забезпечення, власне даних від навмисних та/або випадкових загроз.

Розуміючи важливість питання безпеки, розробники СКБД, у тому числі NewSQL, намагаються вирішити ці питання в рамках своїх систем, не покладаючись на захист через застосунки. Засоби та методи захисту в різних NewSQL системах відрізняються один від одного, однак у тій чи іншій мірі в них, як правило, використовуються зрілі, перевірені часом механізми захисту, що застосовуються у традиційних реляційних СКБД. Так, наприклад, NuoDB подібно до класичних SQL БД підтримує механізми: обмежень цілісності (первинні, унікальний, зовнішні ключі, обмеження not null), надання прав доступу, ролей, ведення журналу бази даних. Крім того, у NuoDB підтримується прозоре шифрування даних (TDE – Transparent Data Encryption), аналогічно використовуваному в Oracle Database, Microsoft SQL Server. TDE гарантує, що дані користувача, що зберігаються в архіві, журналі, резервних копіях, spill-файлах (файли для збереження проміжних даних, коли в пам'яті недостатньо пам'яті для виконання запиту) будуть зашифровані перед записом на диск. Щоб захистити дані, збережені на диску, від несанкціонованого доступу на рівні операційної системи, СКБД SAP HANA також підтримує цей механізм шифрування на відповідному рівні (шифрування тому даних захищає область даних, шифрування журналу повторного виконання (журналу із записом наперед) захищає область журналу на диску). Дані зашифровуються за допомогою алгоритму AES-256-CBC. При цьому, якщо використовується енергонезалежна пам'ять, можна використовувати апаратне шифрування для захисту даних на диску. Коли законний користувач отримує доступ до бази даних SAP HANA за допомогою інтерфейсу клієнта (наприклад, ODBC, JDBC або HTTP), його можливість виконувати операції над об'єктами бази даних визначається наданими йому привілеями. Усі привілеї, надані користувачеві прямо чи опосередковано через ролі, об'єднуються. Це означає, що кожного разу, коли користувач намагається отримати доступ до об'єкта, система перевіряє авторизацію користувача, призначених йому ролей і безпосередньо наданих привілеїв. У SAP HANA використовуються кілька типів привілеїв: системні (system), об'єктні (object), аналітичні (analytic), пакетні (package) та прикладні (application). При цьому привілеї можуть зв'язуватися з роллю (роль може містити будь-яку кількість привілеїв). Як тільки всі запитані привілеї будуть знайдені, система надає доступ. У SAP HANA підтримується механізм маскування даних (data masking), що забезпечує так званий додатковий рівень керування доступом, який можна застосовувати до таблиць та уявлень. Методи анонімізації (anonymization), доступні в SAP HANA, дозволяють отримувати статистично достовірні відомості зі збережених даних, захищаючи при цьому конфіденційність окремих осіб.

Моделі авторизації в CockroachDB використовують особливу концепцію користувача та ролей. CockroachDB не має технічних відмінностей між роллю або користувачем. Так виконання операторів SQL CREATE USER і CREATE ROLE призведе до створення одного й того самого об'єкта з одним винятком: CREATE ROLE буде додано NOLOGIN опція за замовчуванням, що запобігає використанню користувача/ролі для входу в систему. Коли користувач підключається до бази даних або через вбудований клієнт SQL, або через клієнтський драйвер, CockroachDB перевіряє привілеї користувача та ролі для кожного оператора, що виконується. Якщо користувач не має достатніх привілеїв для виконання оператора, CockroachDB видає помилку. Крім привілеїв на: базу даних, функцію, схему, таблицю, послідовність, тип, у CockroachDB є так звані привілеї системного рівня – це особливий вид привілеїв, які застосовуються до всього кластера, що означає, що привілей не прив'язаний до будь-якого конкретного об'єкта у базі даних. Мережевий трафік у CockroachDB між вузлами, а також від клієнтів до вузлів шифрується за допомогою TLS (Transport Layer Security). Шифрування даних вузла на локальному диску забезпечується за допомогою прозорого механізму шифрування. Він дозволяє шифрувати всі файли на диску за допомогою AES у режимі лічильника з розміром ключа 128, 192 або 256 біт [85].

Безпека в LeanXcale досягається за рахунок:

- мережного шифрування, що забезпечує конфіденційність даних (дані зашифровані та доступні для читання тільки клієнту та LeanXcale), автентифікацію повідомлення

(підтверджує, що повідомлення надіслано з LeanXcale або допустимого клієнта), цілісність повідомлень (підтверджує, що жодне з повідомлень не було змінено з моменту відправлення), невідмовність (nonrepudiation – не дозволяє відправникам заперечувати надсилання зашифрованого повідомлення);

- шифрування даних (гарантує конфіденційність даних, що зберігаються на жорстких дисках, завдяки використанню сучасного алгоритму шифрування AES; рекомендується використання AES 256 у режимі GCM – Galois/Counter Mode);

- автентифікації користувача;

- авторизації, що управляє ролями та дозволами для кожної таблиці та користувача в базі даних.

Автентифікація в MariaDB Xpand здійснюється за допомогою облікових записів користувачів бази даних. Облікові записи користувачів бази даних визначаються ім'ям користувача, ім'ям хоста, з якого підключається обліковий запис, та методом автентифікації, налаштованим для облікового запису. MariaDB Xpand підтримує систему керування доступом, аналогічну до тієї, що використовується в MySQL. Привілеї можуть надаватися глобально, на рівні БД або на рівні таблиці. Також для керування доступом до об'єктів бази даних MariaDB Xpand використовує ролі. Тобто є можливість надання привілеїв ролям та відкликання привілеїв у ролей. Користувачеві може призначатися кілька ролей, і користувач може активувати кілька ролей одночасно. Після надання користувачеві ролі він може використовувати всі привілеї, якими володіє дана роль. При наданні ролі іншої ролі, ця роль успадковує всі привілеї, що належать наданої ролі, і користувачі з цією роллю можуть використовувати всі привілеї, що належать обом ролям.

Контроль доступу користувачів та груп до ресурсів Google Cloud Spanner на рівні проєкту, екземпляра Spanner та бази даних Spanner здійснюється за допомогою системи керування обліковими записами та доступом (IAM – Identity and Access Management). Використання IAM дозволяє надавати дозволи (permissions) користувачеві або групі без необхідності змінювати кожен екземпляр Spanner або дозвіл на базу даних окремо. Дозволи дають змогу користувачам виконувати певні дії з ресурсами Spanner. Дозволи не надаються користувачам безпосередньо. Користувачам надаються так звані попередньо визначені / зумовлені ролі (predefined roles) або ролі, що настроюються (custom roles), які мають один або кілька дозволів, пов'язаних з ними. Крім того, Cloud Spanner підтримує деталізований (fine-grained) контроль доступу, який поєднує переваги IAM з традиційним контролем доступу на основі ролей. Деталізований контроль доступу дозволяє налаштувати докладні політики доступу на рівні таблиць і стовпців. Для керування політиками на рівні таблиць та стовпців використовуються відповідні оператори DDL (Data Definition Language) SQL та функції IAM. Комплексна стратегія безпеки Google включає шифрування в стані спокою (зберігання), яке допомагає захистити вміст клієнтів від зловмисників. Зашифровується весь контент клієнтів Google, що зберігається, без будь-яких дій з боку останніх. Усі системи зберігання Google використовують однакову архітектуру шифрування, хоча деталі реалізації відрізняються від системи до системи. У Spanner є три рівні шифрування. Дані в стані спокою розбиваються на фрагменти підфайлів для зберігання і кожен фрагмент шифрується на рівні сховища за допомогою окремого ключа шифрування. Розмір кожного фрагмента може досягати кількох гігабайт. Ключ, який використовується для шифрування даних у блоці, називається ключем шифрування даних (DEK – data encryption key). Два фрагменти не будуть мати однаковий DEK, навіть якщо вони належать одному і тому ж клієнту або зберігаються на одному комп'ютері. Якщо фрагмент даних оновлюється, він шифрується за допомогою нового ключа, а не повторним використанням існуючого ключа. Такий поділ даних, у кожному з яких використовується свій ключ, обмежує ризик потенційної компрометації ключа шифрування даних лише цим блоком. Через великий обсяг ключів у Google та необхідності малої затримки та високої доступності ці ключі зберігаються поруч із даними, які вони шифрують. DEK зашифровуються («обгортаються» – wrapped) за допомогою ключа шифрування ключів (KEK – key en-

cryption key). Нарешті, кожен KEK шифрується ключем шифрування, керованим клієнтом (customer-managed encryption key). Google за допомогою алгоритму AES шифрує дані перед записом їх у систему зберігання БД або на апаратний диск. Шифрування вбудовано у всі системи зберігання. Кожен блок даних має унікальний ідентифікатор. Списки контролю доступу (ACL – access control lists) допомагають гарантувати, що кожен фрагмент може бути розшифрований лише службами Google, які працюють з авторизованими ролями, яким надається доступ лише в даний момент часу. Це обмеження доступу допомагає запобігти доступу до даних без авторизації, зміцнюючи безпеку та конфіденційність даних.

У SingleStore для розмежування доступу використовується метод керування доступом на основі ролей (RBAC – Role Based Access Control). Для цього визначається стандартний перелік таких ролей, можливі зв'язки між користувачами, ролями та групами: роль може мати декілька привілеїв; група може мати кілька ролей; у групі може бути декілька користувачів; користувач може мати кілька ролей; користувач може бути включений до декількох груп; користувачі успадковують дозволи, ролі груп, до яких вони належать. Крім того, у SingleStore підтримується детальний контроль доступу – захист на рівні рядків (Row Level Security). Безпека на рівні рядків у SingleStore досягається за допомогою створення уявлення таблиці зі спеціальним стовпцем ролей. Всі дії з базою даних записуються в журнал аудиту (існує 11 рівнів ведення журналу, які можна розділити на три категорії, кожна з яких має зростаючий рівень деталізації). Специфікація формату шифрування дисків SingleStoreDB сумісна з LUKS (Linux Unified Key Setup). Дані у стані спокою в SingleStoreDB можна захистити за допомогою IBM Guardium Data Encryption. Усю інформацію SingleStore, включаючи файли даних, резервні копії та журнали, можна захистити від несанкціонованого доступу, у тому числі з боку неавторизованих користувачів з правами адміністратора, за допомогою прозорого шифрування CipherTrust Transparent Encryption (CTE) від Thales. Цей процес також відомий як прозоре шифрування бази даних або TDE.

За замовчуванням Volt Active Data (VoltDB) не виконує жодних перевірок безпеки, коли клієнтський застосунок відкриває з'єднання з базою даних або викликає збережену процедуру. Це зручно при розробці та розповсюдженні застосунків у приватній мережі. Однак у загальнодоступних або напівприватних мережах важливо переконатися, що лише відомі клієнтські застосунки взаємодіють із базою даних. При відповідних налаштуваннях безпеки ім'я користувача та пароль, передані клієнтським застосунком, перевіряються сервером на відповідність їх користувачам, визначеним у файлі конфігурації. Якщо клієнтський застосунок передає допустиму пару імені користувача та пароля для облікового запису, термін дії якого ще не минув, з'єднання встановлюється. Коли застосунок викликає збережену процедуру, дозволи перевіряються знову. Якщо схема визначає, що користувачу призначено роль, що має доступ до цієї збереженої процедури, процедура виконується. В іншому випадку застосунку, що викликає, повертається помилка. Volt Active Data підтримує механізм ролей.

MariaDB MaxScale забезпечує корпоративну безпеку, перехоплюючи запити до бази даних до того, як вони досягнуть БД, що дозволяє запобігти розкриттю та пошкодженню даних, а також захистити базу даних від виведення з ладу через зловмисні або випадкові запити. У MariaDB MaxScale є механізм захисту від DoS-атак. Розширені функції безпеки у MariaDB MaxScale можуть відхиляти/блокувати небезпечні, непідтверджені (unapproved) або інші підозрілі запити, а також змінювати результати запитів для захисту даних [86]. Спеціальний фільтр обмеження результатів запобігає випадковим або зловмисним запитам, які роблять базу даних недоступною або розкривають великі обсяги даних, обмежуючи кількість результатів, які може повернути запит (наприклад, щоб запит не повертав усі десять мільйонів рядків у таблиці). Це не тільки зменшує трафік, а й запобігає розкриттю значних обсягів даних зловмисникові, який намагається різними способами їх отримати. Для захисту конфіденційної та/або особистої інформації від розкриття використовується механізм динамічного маскування. Для безпечних з'єднань із застосунками та базами даних MariaDB MaxScale підтримує SSL/TLS. Крім того, MariaDB MaxScale підтримує PAM (Pluggable Authentication

Modules) та GSSAPI (основною використовуваною реалізацією механізму GSSAPI (Generic Security Service Application Program Interface) є Kerberos) для автентифікації. MariaDB MaxScale може бути налаштований на використання проксі-протоколу передачі інформації про клієнта на сервер MariaDB для подальшого спрощення керування користувачами.

Amazon Aurora підтримує кілька способів автентифікації користувачів бази даних: автентифікацію за паролем (доступна за замовчанням для кластерів БД); IAM автентифікацію (для Aurora MySQL, Aurora PostgreSQL); Kerberos автентифікацію для одного кластера БД (для Aurora PostgreSQL). Конкретний користувач може увійти до бази даних, використовуючи лише один метод автентифікації. Amazon Aurora може шифрувати кластери БД. Для цього використовується алгоритм шифрування AES-256. Зашифровуватись можуть дані базового сховища кластерів БД, його автоматичні резервні копії, репліки читання та моментальні знімки. У процесі роботи Amazon Aurora виконує автентифікацію доступу і прозоро з мінімальним впливом на продуктивність розшифровує дані. Однак слід пам'ятати, що шифрування існуючого незашифрованого екземпляра Aurora на даний момент не підтримується. Тому, щоб використовувати шифрування Amazon Aurora для існуючої незашифрованої бази даних, потрібно спочатку створити новий екземпляр БД із «включеним» шифруванням, після чого перенести до нього відповідні дані. Aurora інтегрується з Amazon GuardDuty для виявлення потенційних загроз даним, що зберігаються в базах даних Aurora. GuardDuty RDS Protection створює профілі та відстежує дії по входу в систему та нові бази даних в обліковому записі користувача, крім того, він використовує спеціалізовані моделі машинного навчання для виявлення підозрілих входів до бази даних Aurora. Для управління доступом до даних в Amazon Aurora підтримується механізм так званих груп безпеки (контролюють доступ трафіку в кластер БД і з нього; в групі безпеки можна вказати до 20 правил), привілеїв та ролей.

У табл. 1 наведено зведені дані про деякі характеристики відомих систем NewSQL.

Якщо говорити в цілому про відмінності між традиційними реляційними, NoSQL та NewSQL базами даних для масштабованих рішень, то в табл. 2 наведено деякі важливі їх відмітні характеристики.

Узагальнюючи все сказане, а також деякі висновки, зроблені іншими авторами [87], слід зазначити, що основними перевагами баз даних NewSQL є:

– забезпечення більш високої узгодженості даних (підтримка ACID-транзакцій);

– можливість використання відомих, перевірених часом, операторів SQL та стандартного інструментарію;

– багатші можливості застосування методів аналізу з використанням SQL та розширень;

– кластеризація у стилі NoSQL з використанням більш традиційних моделей даних та запитів.

Крім того, можна також додати, що NewSQL системи дозволяють виконувати гібридну транзакційно-аналітичну обробку (HTAP), мета якої – виконання застосунків OLAP та OLTP на одних і тих самих даних. Це дозволяє проводити аналіз оперативних даних у реальному масштабі часу без традиційного поділу оперативної бази даних та сховища даних, уникаючи складнощів, пов'язаних з ETL (Extract, Transform, Load – витяг, перетворення, завантаження).

Основні недоліки NewSQL баз даних:

– архітектури в оперативній пам'яті можуть не підходити для обсягів, що перевищують кілька терабайт;

– підтримують лише частковий доступ до багатого інструментарію традиційних систем SQL.

ISSN 0485-8972 Radiotekhnika No. 211 (2022)
eISSN 2786-5525

Таблиця 1

Категорія	СКБД	Рік випуску	Поточний випуск/дата	Архітектура зберігання даних	Тип	Розбиття	Керування паралельною обробкою	Реплікація	Короткий опис
Системи з використанням нової архітектури	**MariaDB Xpand** (раніше **Clustrix)**	Clustrix, 2006	MariaDB Xpand: v6.1.0, 02.2023	Гібридна: у пам'яті та на SSD	OLTP	Підтримує	MVCC+2PL	active-active, active-passive	Розподілена БД SQL, яка ефективно масштабується для сучасних веб-застосунків з великим робочим навантаженням, які потребують суворої узгодженості та цілісності даних.
	CockroachDB	2014	v21.1.2, 06.2021	Дискове сховище	OLTP	Підтримує	MVCC	active-passive	Розподілена БД SQL з відкритим вихідним кодом, побудована на транзакційному та строго узгодженому сховищі ключ-значення. Задля узгодженості використовує алгоритм розподіленого консенсусу Raft.
	Google Cloud Spanner	2012 (Spanner) / 2017 (Cloud Spanner)	Немає даних	Дискове сховище	OLTP	Підтримує	MVCC+2PL	active-passive	Реплікована за глобальною мережею СКБД без спільного використання ресурсів, що використовує спеціальне обладнання (GPS, атомний годинник) для генерації позначок часу/часових міток (для високоточної синхронізації годинника). Широко використовується як СКБД OLTP для структурованих даних у Google та загальнодоступна у бета-версії як Cloud Spanner на хмарній платформі Google (GCP, Google Cloud Platform).
	Volt Active Data (раніше **H-Store/ VoltDB)**	2007 (H-Store) / 2008 (VoltDB)	v.12.1, 12.2022	У пам'яті (in-memory). РККТ – на SSD або HDD	OLTP	Підтримує	TO	active-active, active-passive	Розподілена база даних із сегментуванням та реплікацією даних. Є як комерційна версія, так і версія спільноти з відкритим вихідним кодом. Орієнтована на певний сегмент бізнес-обчислень, а саме на швидку обробку великих потоків даних.
	HyPer	2010	Немає даних	У пам'яті (in-memory)	HTAP: OLTP+ OLAP	Підтримує	MVCC	active-passive	База даних в оперативній пам'яті, метою якої є досягнення високої продуктивності як для робочого навантаження OLTP, так і для OLAP (HTAP – Hybrid transaction/analytical processing).
	SingleStore (раніше **MemSQL)**	2012	SingleStore: v8.0 12.2022	Rowstore – у пам'яті; Columnstore – на диску	HTAP	Підтримує	MVCC	active-passive	Розподілена БД у пам'яті, призначена як для транзакційних, так і для аналітичних робочих навантажень. Підтримує дротовий протокол MySQL. Дані для таблиць rowstore (зберігають інформацію у форматі рядків) зберігаються у пам'яті. Дані таблиць columnstore (сховище стовпців) зберігаються на диску. Миттєві знімки та журнали транзакцій зберігаються на диску.
	NuoDB	2013	v5.0.1, 2020	Гібридна: TE – у пам'яті, SM – на диску	OLTP	Підтримує	MVCC	active-passive	Розподілена система керування базами даних, що використовує архітектуру у стилі мікросервісів, яка поділяє обробку на три рівні: рівень управління, рівень транзакцій та рівень зберігання. Механізми транзакцій (TE) надають кеш, який є базою даних у пам'яті для швидкої обробки транзакцій. Диспетчери сховища (SM) забезпечують довговічність.

Продовження табл. 1

Категорія	СКБД	Рік випуску	Поточний випуск/дата	Архітектура зберігання даних	Тип	Розбиття	Керування паралельною обробкою	Реплікація	Короткий опис
	LeanXcale	2015	Немає даних	Дискове сховище	HTAP	Підтримує	MVCC LeanXcale	active-active	Швидка та масштабована БД, яка поєднує в собі характеристики SQL та NoSQL. Вона створена для прийому великих потоків даних у реальному масштабі часу, а також для можливості паралельного виконання різноманітних запитів для будь-якого використання.
	SAP HANA	2010	v.2.0 SPS06, 03.2022	У пам'яті (in-memory). РККТ – на дисках	HTAP	Підтримує	MVCC	active-passive	Система підтримує зберігання таблиць як по стовпцям, так і по рядках (у першому випадку для транзакційних навантажень, у другому – для аналітичних). SAP HANA активно використовує NVM.
ПЗ проміжного рівня	**MariaDB MaxScale**	2015	(v22.08.3, 12.2022); (v2.5.24, 01.2023); (v6.4.4, 11.2022)	-	HTAP	Підтримує	MVCC+2PL	active-active, active-passive	Проксі БД, маршрутизатор і балансувальник навантаження, який розширює можливості високої доступності, масштабованості та безпеки сервера MariaDB, відокремлюючи його від базової інфраструктури БД, та спрощує розробку застосунків. MariaDB MaxScale може надсилати різні запити до різних екземплярів БД.
	ScaleArc	2009	v.3.12 GA, 10.2017	-	HTAP	Підтримує	Mixed	active-passive	ScaleArc – балансувальник навантаження БД. Він дозволяє адміністраторам БД створювати високодоступні, масштабовані та прості в керуванні, обслуговуванні та міграції розгортання бази даних. Програмне забезпечення ScaleArc доступне для баз даних MySQL, SQL Server та Oracle. ScaleArc працює з Microsoft SQL Server та MySQL як локальне рішення, у хмарі для відповідних PaaS або як рішення DBaaS, включаючи Amazon RDS або AzureSQL.
БД як послуга	**Amazon Aurora**	2014	Aurora MySQL 3, 11.2021; Aurora PostgreSQL 14.3, 06.2022	Дискове сховище	OLTP	Не підтримує	MVCC	active-passive	Amazon Aurora – це реляційна база даних, що пропонується як сервіс в Amazon AWS. Заснована на версії MySQL з відкритим вихідним кодом, це комерційна база даних, заявлена як сумісна з MySQL та PostgreSQL та забезпечує високу пропускну спроможність.

Таблиця 2

Характеристика	RDBMS	NoSQL	NewSQL
Схема	Набір таблиць	Без схеми	Обидва підходи
SQL	Підтримується	Залежить від системи	Підтримується
ACID	Підтримується	Не підтримується	Підтримується
OLTP	Підтримується не повною мірою	Підтримується	Повна підтримка
Підтримка масштабування: вертикального / горизонтального	Так / Є реалізації для деяких СКБД (Oracle, SQL Server)	Так / Так	Так / Так
Можливість побудови розподілених систем	Так	Так	Так
Безпека	Висока	Низька	Висока
Архітектура зберігання даних	Дискове сховище / В пам'яті	Дискове сховище / В пам'яті	Дискове сховище / В пам'яті
Вплив збільшення розміру даних на продуктивність	Повільно	Швидко	Дуже швидко
Складність запитів	Низька	Висока	Висока
Робоче навантаження	OLTP / OLAP	OLTP	OLTP, OLAP, HTAP
Популярність (підтримка спільноти)	Величезна	Зростаюча	Зростаюча

Висновки

1. Поява NewSQL систем є відображенням об'єктивної потреби в масштабованих СКБД з підтримкою SQL та ACID-транзакцій, тобто таких систем, які прагнуть досягти продуктивності, порівнянної з NoSQL рішеннями, зберігаючи при цьому гарантії узгодженості даних. При цьому, безумовно, основною перевагою NewSQL систем є підтримка ACID-транзакцій, що уможливлює їх застосування там, де NoSQL рішення непридатні. Крім того, підтримка SQL дозволяє використовувати накопичений досвід та інструментарій для роботи з традиційними базами даних. Хоча при цьому слід враховувати, що створення схеми та оптимізація SQL запитів для виконання на кластері можуть ускладнити розробку застосунків.

2. NewSQL системи з новими архітектурами принципово відрізняються від традиційних SQL-орієнтованих СКБД початковою підтримкою розподіленої архітектури. Ці системи здатні добре масштабуватися горизонтально і забезпечувати високу продуктивність за певних типів транзакцій, що зачіпають невелику кількість вузлів. На таких транзакціях NewSQL системи наближаються за масштабованістю та продуктивністю до NoSQL рішень, але при цьому зберігаючи підтримку ACID та SQL. Крім того, NewSQL СКБД з новими архітектурами прагнуть підвищити продуктивність за рахунок оптимізації швидкості доступу до даних, у тому числі за рахунок зберігання бази даних повністю в оперативній пам'яті та ігнорування блокувань. При цьому, швидше за все, коли ширше використовуватимуться технології енергонезалежної пам'яті знадобляться подальші зміни в конструкції підсистем (в даний час це відносно нова галузь досліджень, що заслуговує на особливу увагу).

3. Використання програмного забезпечення проміжного рівня дозволяє прозоро розділяти дані між декількома екземплярами БД, знімаючи необхідність реалізації шардингу на стороні застосунку. Розробникам не потрібно вносити будь-які зміни до свого застосунку, щоб використовувати нову сегментовану базу даних. Для сумісності з конкретною СКБД проміжне програмне забезпечення має підтримувати відповідний протокол обміну. Продуктивність таких рішень загалом нижча, ніж, наприклад, систем з новими архітектурами, у тому числі через надмірність планування та оптимізації запитів до фрагментів даних, розміщених на окремих вузлах, для складних запитів. Хоча з іншого боку, це дозволяє кожному вузлу застосовувати власні локальні оптимізації для кожного запиту.

4. Певні NewSQL СКБД дозволяють проводити гібридну транзакційно-аналітичну обробку. Тобто дозволяють виконувати аналіз оперативних даних у реальному масштабі часу без традиційного поділу оперативної бази даних та сховища даних, уникаючи складнощів, пов'язаних із процесом ETL.

5. Через те, що найчастіше продуктивність є головним пріоритетом, вважається, що бази даних NewSQL мають більше вразливих місць у безпеці, ніж традиційні реляційні бази даних. Однак для більш обґрунтованого висновку все ж таки потрібні додаткові всебічні дослідження цієї проблеми.

6. Важливим висновком нашого аналізу вважатиметься те, що системи баз даних NewSQL не є принциповим відходом від існуючих принципів побудови БД. Вони швидше за все є наступним витком спіралі у розвитку технологій баз даних. Зокрема, існує низка рішень для масштабування класичних SQL-орієнтованих СКБД, що допомагають масштабувати існуючі застосунки без значних змін логіки роботи з даними. У більш довгостроковій перспективі можна вважати, що відбудеться конвергенція функцій у основних класах систем керування даними. Швидше за все, всі ключові системи будуть підтримувати ту або іншу форму реляційної моделі та SQL, а також операції OLTP та запити OLAP одночасно подібно до HTAP-систем.

Список літератури:

1. Abadi D., Ailamaki A., Andersen D., Bailis P., Balazinska M., Bernstein P., Boncz P., Chaudhuri S., et al // The Seattle Report on Database Research. ACM SIGMOD Record. 2020. 48. P. 44 – 53. https://doi.org/10.1145/3385658.3385668.
2. Gudivada V. N., Rao D., Raghavan V. V. Renaissance in database management: navigating the landscape of candidate systems // Computer. 2016. 49(4). P. 31–42. https://doi.org/10.1109/MC.2016.115.
3. Sadalage P. J., Fowler M. NoSQL Distilled A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley Professional, 2012. 188 p.
4. Using Oracle Sharding. Oracle Sharding Overview. URL: https://docs.oracle.com/en/database/oracle/oracle-database/19/shard/sharding-overview.html#GUID-0F39B1FB-DCF9-4C8A-A2EA-88705B90C5BF. (дата звернення: 17.02.2023).
5. Shute J., Vingralek R., Samwel B., Handy B., Whipkey C., Rollins E., Oancea M., Littlefield K., Menestrina D., Ellner S., Cieslewicz J., Rae I., Stancescu T., Apte H. F1: A distributed SQL database that scales // Proceedings of the 39th International Conference on Very Large Data Bases (VLDB) Endowment. 2013. 6(11). 1068 – 1079.
6. Pavlo A., Aslett M. What's really new with NewSQL? // ACM Sigmod Record. 2016. 45(2). P. 45 – 55. https://doi.org/10.1145/3003665.3003674.
7. NoSQL. URL: https://hostingdata.co.uk/nosql-database/ (дата звернення: 17.02.2023).
8. Corbett, J. C., Dean, J., Epstein, M., Fikes, A., Frost, C., Furman, J. J., Ghemawat S., Gubarev A., Heiser C., Hochschild P., Hsieh W., Kanthak S., Kogan E., Li H., Lloyd A., Melnik S., Mwaura D., Nagle D., Quinlan S., Rao R., Rolig L., Saito Y., Szymaniak M., Taylor C., Wang R., Woodford D. Spanner: Google's globally distributed database // ACM Transactions on Computer Systems (TOCS). 2013. 31(3). P. 1 – 22. https://doi.org/10.1145/2491245.
9. Özsu M. T., Valduriez P. Principles of Distributed Database Systems. Fourth Edition. Springer Cham, 2020. 674 p.
10. Aslett M. What we talk about when we talk about NewSQL. URL: http://blogs.the451group.com/information_management/2011/04/06/what-we-talk-about-when-we-talk-about-newsql/ (дата звернення: 17.02.2023).
11. Valduriez P., Jiménez-Peris R., Özsu M. T. Distributed database systems: The case for NewSQL // Hameurlain, A., Tjoa, A.M. (eds) Transactions on Large-Scale Data- and Knowledge-Centered Systems XLVIII. Lecture Notes in Computer Science. Vol 12670. Berlin, Heidelberg: Springer Berlin Heidelberg, 2021. P. 1 – 15. https://doi.org/10.1007/978-3-662-63519-3_1.
12. Moniruzzaman A. B. M. NewSQL: Towards next-generation scalable RDBMS for online transaction processing (OLTP) for Big Data management // International Journal of Database Theory and Application. 2014. 7(6) P. 121 – 130 http://dx.doi.org/10.14257/ijdta.2014.7.6.11.
13. Stonebraker M. The case for shared nothing // IEEE Database Eng. Bull. 1986. 9(1). P. 4 – 9.
14. Duggirala S. NewSQL databases and scalable in-memory analytics // Advances in Computers. Elsevier, 2018. 109. P. 49 – 76. https://doi.org/10.1016/bs.adcom.2018.01.004.
15. MariaDB Xpand. URL: https://mariadb.com/products/enterprise/xpand/ (дата звернення: 17.02.2023).
16. MariaDB. MariaDB Acquires Clustrix Adding Distributed Database Technology. URL: https://mariadb.com/newsroom/press-releases/mariadb-acquires-clustrix-adding-distributed-database-technology/ (дата звернення: 17.02.2023).

17. Namuag P. An Overview of MariaDB Xpand (formerly ClustrixDB). URL: https://severalnines.com/blog/overview-mariadb-xpand-formerly-clustrixdb/ (дата звернення: 17.02.2023).
18. Clustrix. URL: https://dbdb.io/db/clustrix (дата звернення: 17.02.2023).
19. MariaDB Xpand. URL: https://mariadb.com/docs/xpand/products/mariadb-xpand/ (дата звернення: 17.02.2023).
20. CockroachDB. URL: www.cockroachlabs.com (дата звернення: 17.02.2023).
21. Cloud Spanner. URL: https://cloud.google.com/spanner/ (дата звернення: 17.02.2023).
22. Bacon D. F., Bales N., Bruno N., Cooper B. F., Dickinson A., Fikes A., Fraser C., Gubarev A., Joshi M., Kogan E., Lloyd A., Melnik S., Rao R., Shue D., Taylor C., Holst M. H., Woodford D. Spanner: Becoming a SQL system // Proceedings of the 2017 ACM International Conference on Management of Data (SIGMOD '17). Association for Computing Machinery, New York, NY, USA, 2017. P. 331 – 343. https://doi.org/10.1145/3035918.3056103.
23. HyPer. URL: https://hyper-db.de/ (дата звернення: 17.02.2023).
24. SingleStore. URL: https://www.singlestore.com/ (дата звернення: 17.02.2023).
25. NuoDB. URL: https://www.nuodb.com. (дата звернення: 17.02.2023).
26. SAP HANA Cloud. URL: www.sap.com/products/hana.html (дата звернення: 17.02.2023).
27. Sikka V., Färber F., Lehner W., Cha S. K., Peh T., Bornhövd C. Efficient transaction processing in SAP HANA database: the end of a column store myth // Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. 2012. P. 731 – 742).
28. H-Store. URL: https://hstore.cs.brown.edu/ (дата звернення: 17.02.2023).
29. Volt Active Data. URL: http://voltactivedata.com/ (дата звернення: 17.02.2023).
30. Simborg M. Introducing Volt Active Data. URL: https://www.voltactivedata.com/blog/2022/02/introducing-volt-active-data/ (дата звернення: 17.02.2023).
31. LeanXcale. URL: https://www.leanxcale.com/ (дата звернення: 17.02.2023).
32. Van Steen M., Tanenbaum A. S. Distributed systems. Third edition. Pearson Education, Inc. 2017. 596 p.
33. Gazis A., Katsiri E. Middleware 101: What to know now and for the future // Queue. 2022. 20(1). P. 10 – 23. https://doi.org/10.1145/3526211.
34. Harizopoulos S., Abadi D. J., Madden S., Stonebraker M. OLTP through the looking glass, and what we found there // Making Databases Work: the Pragmatic Wisdom of Michael Stonebraker. Association for Computing Machinery and Morgan & Claypool. 2018. P. 409–439. https://doi.org/10.1145/3226595.3226635.
35. MariaDB MaxScale. URL: https://mariadb.com/products/enterprise/components/#maxscale (дата звернення: 17.02.2023).
36. ScaleArc. URL: www.devgraph.com/scalearc (дата звернення: 17.02.2023).
37. Bernstein P. A., Cseri I., Dani N., Ellis N., Kalhan A., Kakivaya G., Lomet D. B., Manne R., Novik L., Talius T. Adapting Microsoft SQL server for cloud computing // 2011 IEEE 27th International Conference on Data Engineering, 2011. P. 1255 – 1263. https://doi.org/10.1109/ICDE.2011.5767935.
38. Amazon Aurora. https://aws.amazon.com/rds/aurora (дата звернення: 17.02.2023).
39. Connolly T. M., Begg C. E. Database systems: a practical approach to design, implementation, and management. Sixth edition. Harlow, Essex, England: Pearson Education Limited, 2015. 1329 p.
40. Kleppmann M. Designing data-intensive applications: The big ideas behind reliable, scalable, and maintainable systems. O'Reilly Media, Inc., 2017. 590 p.
41. Gray J., Reuter A. Transaction processing: concepts and techniques. Elsevier, 1992. 1070 p.
42. S. Kimball. Living without atomic clocks. URL: https://www.cockroachlabs.com/blog/living-without-atomic-clocks/ (дата звернення: 17.02.2023).
43. Spanner: TrueTime and external consistency. URL: https://cloud.google.com/spanner/docs/true-time-external-consistency (дата звернення: 17.02.2023).
44. TimeTools. What is the GPS Clock? URL: https://timetoolsltd.com/gps/what-is-the-gps-clock/ (дата звернення: 17.02.2023).
45. Kimball S., Sharif I. Living without atomic clocks. URL: https://www.cockroachlabs.com/blog/living-without-atomic-clocks/. (дата звернення: 17.02.2023).
46. Lamport L. The implementation of reliable distributed multiprocess systems. Computer Networks (1976). 1978. 2(2). P. 95–114. https://doi.org/10.1016/0376-5075(78)90045-4.
47. The Tandem Database Group. NonStop SQL: A distributed, high-performance, high-availability implementation of SQL // Gawlick, D., Haynie, M., Reuter, A. (eds) High Performance Transaction Systems. HPTS 1987. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg. 2005. Vol. 359. P. 60 – 104. https://doi.org/10.1007/3-540-51085-0_43.
48. Malviya N., Weisberg A., Madden S., Stonebraker M. Rethinking main memory OLTP recovery. In 2014 IEEE 30th International Conference on Data Engineering, Chicago, IL, USA, IEEE. 2014. P. 604 – 615, https://doi.org/10.1109/ICDE.2014.6816685.
49. Schwartz B., Zaitsev P., Tkachenko V. High performance MySQL: optimization, backups, and replication. Third Edition. O'Reilly Media, Inc., 2012. 826 p.
50. Harrington J. L. Relational database design and implementation. 4th edition. Morgan Kaufmann, 2016. 712 p.

51.Navathe S., Ceri S., Wiederhold G., Dou J. Vertical partitioning algorithms for database design // ACM Transactions on Database Systems (TODS). 1984. 9(4). P. 680–710. https://doi.org/10.1145/1994.2209.

52. Curino C., Jones E., Zhang Y., Madden S. Schism: a workload-driven approach to database replication and partitioning // Proceedings of the VLDB Endowment. 2010. 3(1-2). P. 48 – 57. https://doi.org/10.14778/1920841.1920853.

53. Law Insider. Legal Definitions Dictionary. URL: https://www.lawinsider.com/dictionary/transaction-engine (дата звернення: 17.02.2023).

54. Elmore A. J., Arora V., Taft R., Pavlo A., Agrawal D., Abbadi A. E. Squall: Fine-grained live reconfiguration for partitioned main memory databases // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. 2015. P. 299–313. https://doi.org/10.1145/2723372.2723726.

55. Serafini M., Mansour E., Aboulnaga A., Salem K., Rafiq T., Minhas U. F. Accordion: Elastic scalability for database systems supporting distributed transactions // Proceedings of the VLDB Endowment. 2014. 7(12). P. 1035 –1046. https://doi.org/10.14778/2732977.2732979.

56. Ismail W., Muhammed A., Abdullah Z. H., Radman A., Hendradi R., Afandi R. R. A Survey of NewSQL DBMSs focusing on Taxonomy, Comparison and Open Issues. Journal of Computer Science & Computational Mathematics. 2021. P. 87-95. https://doi.org/10.20967/jcscm.2021.04.002.

57. Rothnie J. B., Bernstein P. A., Fox S., Goodman N., Hammer M., Landers T. A., Reeve C., Shipman D. W., Wong E. Introduction to a system for distributed databases (SDD-1) // ACM Transactions on Database Systems (TODS). 1980. 5(1), P. 1–17. https://doi.org/10.1145/320128.320129.

58. Williams R., Daniels D., Haas L., Lapis G., Lindsay B. G., Ng, P., Obermarck R., Selinger P., Walker A., Wilms P., Yost R. R*: An overview of the architecture // IBM Thomas J. Watson Research Division. 1981. P. 329 – 347.

59. Epstein R., Stonebraker M., Wong E. Distributed query processing in a relational data base system // Proceedings of the 1978 ACM SIGMOD international conference on management of data. (SIGMOD '78). Association for Computing Machinery, New York, NY, USA. 1978. P. 169 – 180. https://doi.org/10.1145/509252.509292.

60. DeWitt D. J., Katz R. H., Olken F., Shapiro L. D., Stonebraker M. R., Wood D. A. Implementation techniques for main memory database systems // Proceedings of the 1984 ACM SIGMOD international conference on management of data. 1984. P. 1 – 8. https://doi.org/10.1145/602259.602261.

61. Faerber F., Kemper A., Larson,P. A., Levandoski J., Neumann T., Pavlo A. Main memory database systems // Foundations and Trends in Databases. 2017. 8(1 – 2), P. 1 – 130.

62. Gawlick D., Kinkade D. Varieties of concurrency control in IMS/VS fast path // IEEE Database Eng. Bull. 1985. 8(2). P. 3 – 10.

63. Lehman T. J., Carey M. J. A Study of Index Structures for Main Memory Database Management Systems // Proceedings of the Twelfth International Conference on Very Large Data Bases, VLDB. 1985. P. 294 – 303.

64. Lehman T. J., Carey M. J. Query processing in main memory database management systems // Proceedings of the 1986 ACM SIGMOD international conference on Management of data. 1986. P. 239 – 250. https://doi.org/10.1145/16894.16878.

65. Lehman T. J., Carey M. J. A recovery algorithm for a high-performance memory-resident database system // ACM SIGMOD Record. 1987. 16(3). P. 104 – 117. https://doi.org/10.1145/38714.38730.

66. Kersten M.L., Apers P.M.G., Houtsma M.A.W., van KuykE.J.A., van de Weg R.L.W. A Distributed, Main-Memory Database Machine // Kitsuregawa M., Tanaka H. (eds) Database Machines and Knowledge Base Machines. The Kluwer International Series in Engineering and Computer Science. Springer, Boston, MA. 1988. Vol. 43. P. 353 – 369. https://doi.org/10.1007/978-1-4613-1679-4_26.

67. Altibase. URL: https://www.altibase.com (дата звернення: 17.02.2023).

68. TimesTen: Fastest OLTP Database, Ultra High Availability, Elastic Scalability. URL: https://www.oracle.com/database/technologies/related/timesten.html (дата звернення: 17.02.2023).

69. Cambridge Dictionary. URL: https://dictionary.cambridge.org/dictionary/english/workload. (дата звернення: 17.02.2023).

70. HPE Nimble Storage Deployment Considerations for Microsoft SQL Server. OLTP Workloads. URL: https://infosight.hpe.com/InfoSight/media/cms/active/public/tmg_HPE_Nimble_Storage_Deployment_Considerations_for_Microsoft_SQL_Server_doc_version_family.whz/xpm1491839725334.html (дата звернення: 17.02.2023).

71. DeBrabant J., Pavlo A., Tu S., Stonebraker M., Zdonik S. Anti-caching: A new approach to database management system architecture // Proceedings of the VLDB Endowment. 2013. 6(14). P. 1942 – 1953. https://doi.org/10.14778/2556549.2556575.

72. Arulraj J., Pavlo A., Dulloor S. R. Let's talk about storage & recovery methods for non-volatile memory database systems // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data (SIGMOD '15). Association for Computing Machinery, New York, NY, USA. 2015. P. 707 – 722. https://doi.org/10.1145/2723372.2749441.

73. Andrei M., Lemke C., Radestock G., Schulze R., Thiel C., Blanco R., Meghlan A., Sharique M., Seifert S., Vishnoi S., Booss D., Peh T., Schreter I., Thesing W., Wagle M., Willhalm T. SAP HANA adoption of non-volatile memory // Proceedings of the VLDB Endowment. 2017. 10(12). P. 1754 – 1765. https://doi.org/10.14778/3137765.3137780.

74. Intel Optane Persistent Memory and SAP HANA Platform Configuration. Technology overview and deployment guidelines for using Intel Optane persistent memory with SAP HANA. Configuration Guide. 2019. URL: https://cdrdv2-public.intel.com/753738/sap-hana-and-intel-optane-configuration-guide.pdf (дата звернення: 17.02.2023).

75. Intel Optane Persistent Memory. URL: https://www.intel.com/content/www/us/en/products/docs/memory-storage/optane-persistent-memory/overview.html (дата звернення: 17.02.2023).

76. Kroenke D. M., Auer D. J., Yoder R. C., Vandenberg S. L. Database processing fundamentals, design, and implementation. 15th edition. Pearson. 2018. 648 p.

77. Taft R., Sharif I., Matei A., VanBenschoten N., Lewis, J., Grieger, T., Niemi K., Woods A., Birzin A., Poss R., Bardea P., Ranade A., Darnell B., Gruneir B., Jaffray J., Zhang L., Mattis P. Cockroachdb: The resilient geo-distributed SQL database // Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 2020. P. 1493 – 1509. https://doi.org/10.1145/3318464.3386134.

78. Garcia-Molina H., Salem K. Main memory database systems: An overview // IEEE Transactions on knowledge and data engineering. 1992. 4(6). P. 509–516. https://doi.org/10.1109/69.180602.

79. Google Cloud Spanner. Secondary indexes. URL: https://cloud.google.com/spanner/docs/secondary-indexes (дата звернення: 17.02.2023).

80. SAP HANA Performance Guide for Developers. Secondary Indexes. URL: https://help.sap.com/docs/SAP_HANA_PLATFORM/9de0171a6027400bb3b9bee385222eff/3441acf7dcf64e169ba94121acaf2350.html?version=2.0.04&locale=en-US (дата звернення: 17.02.2023).

81. Fitzpatrick B. Distributed caching with Memcached // Linux journal. 2004. 2004(124). P. 5.

82. Mohan C., Haderle D., Lindsay B., Pirahesh H., Schwarz P. ARIES: A transaction recovery method supporting fine-granularity locking and partial rollbacks using write-ahead logging // ACM Transactions on Database Systems (TODS). 1992. 17(1). P. 94 – 162. https://doi.org/10.1145/128765.128770.

83. Using VoltDB. V.12.1. URL: https://docs.voltdb.com/UsingVoltDB/ (дата звернення: 17.02.2023).

84. Ren K., Diamond T., Abadi D. J., Thomson A. Low-overhead asynchronous checkpointing in main-memory database systems // Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA. 2016. P. 1539 – 1551. https://doi.org/10.1145/2882903.2915966.

85. Reid R. Practical CockroachDB: Building Fault-Tolerant Distributed SQL Databases. 1st ed. Apress. 2022. 254 p.

86. MariaDB MaxScale technical brief. Enterprise security. URL: https://mariadb.com/wp-content/uploads/2019/09/mariadb-maxscale-security_datasheet_1041.pdf (дата звернення: 17.02.2023).

87. Khasawneh T. N., AL-Sahlee M. H., Safia A. A. SQL, NewSQL, and NoSQL databases: A comparative survey // 2020 11th International Conference on Information and Communication Systems (ICICS), Irbid, Jordan, IEEE. 2020. P. 013 – 021. https://doi.org/10.1109/ICICS49469.2020.239513.

Надійшла до редколегії 20.11.2022

Відомості про авторів:

Есін Віталій Іванович – д-р техн. наук, Харківський національний університет імені В.Н. Каразіна, професор кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Україна; e-mail: v.i.yesin@karazin.ua; ORCID: https://orcid.org/0000-0003-1977-7269

Вілігура Владислав Вікторович – Харківський національний університет імені В.Н. Каразіна, аспірант кафедри безпеки інформаційних систем і технологій, факультет комп'ютерних наук; Україна; e-mail: viligura93@gmail.com; ORCID: https://orcid.org/0000-0002-1137-2382