УДК 004.657

А. А. Єфіменко, В. Н. Ковальчук, Г. О. Мішин, І. І. Сугоняк

МОДЕЛЬ ДИСПЕТЧЕРИЗАЦІЇ ПОТОКІВ ДАНИХ ДЛЯ ВИСОКОНАВАНТАЖЕНИХ ВЕБ-СИСТЕМ

У статті запропоновано модель оптимального розподілення даних між кластерами (серверами) в умовах високонавантаженої системи та описано побудовану на їх основі прикладну веб-орієнтовану систему. Як проміжний сервер використано нереляційну систему управління базами даних MongoDB, для якої реалізовано алгоритм збору статистики та побудови ключа розподілу даних. Для отримання інформації щодо реєстрації подій у реальній діючій системі було запропоновано використати чотирикрокову процедуру проксингу трафіка. Після збору запитів користувачів та трансформації їх у зручну форму розроблено алгоритм для побудови статистики. Було проведено три незалежні тести з такими моделями: без розподілення даних за шардами; стандартне розподілення за основним ключем; розподілення за створеним ключем згідно з розробленою моделлю. Вибрано 80 тисяч запитів, однакових для кожної ітерації. Ефективність шардингу даних на основі ключа розподілу підтверджена тестовими експериментами.

Ключові слова: *оптимізація запитів, шардинг, високонавантажені системи, SQL Server, MongoDB.*

Постановка проблеми в загальному вигляді. Одним із найбільш актуальних питань на сьогодні є оптимізація зберігання та обробки даних у високонавантажених веб-системах, що забезпечують інформаційний обмін у всіх сферах життя сучасного суспільства, починаючи з управління державними підприємствами, закінчуючи складною системою регулювання світовою мережею закладів із системами моніторингу “наживо” та великими обсягами звітної інформації.

У ході дослідження вирішується існуюча проблема оптимального розподілення даних у сховищах високонавантажених веб-систем, зумовлена зростанням цінності та важливості інформації у свідомості людини.

Аналіз останніх досліджень і публікацій. Оптимізація запитів до баз та сховищ даних є дуже обговорюваним у науковій спільноті питанням. Дана проблематика розглядається в працях визначних закордонних дослідників-практиків: К. Дейта (C. Date), Дж. Герке (J. Gehrcke), Е. Бревера (E. Brewer) [1], М. Еббота (Martin L. Abbott), М. Фішера (Michael T. Fisher) та багато інших. Поряд з академічними дослідженнями друкуються публікації компаній та фахівців-практиків, присвячені розв’язанню цього завдання в рамках окремих систем управління базами даних (СУБД).

Формулювання завдання дослідження. Метою дослідження є розробка моделей і методів оптимального розподілення даних між кластерами (серверами) в умовах високонавантаженої системи та побудова на їх основі прикладної системи для оптимального розподілення даних, що обумовило такі завдання: побудувати моделі

розподілення даних з урахуванням доступних потужностей системи та можливостей їх використання; розробити алгоритм розбиття даних на кластери (сервери); провести тестування алгоритму в умовах навантаження та побудувати графіки його роботи.

Виклад основного матеріалу. Для систем баз даних з великими їх обсягами або програм з високою пропускною здатністю може не вистачати потужності одного сервера. Існує два методи для вирішення проблеми зростань обсягів: вертикальне і горизонтальне масштабування. Для розподілення даних та прискорення часу їх отримання використовують сегментування.

Сегментування, шардування (Sharding) – це спосіб розподілу даних між декількома серверами (Shards – шардами). СУБД MongoDB використовує сегментування для підтримки розгортання баз з дуже великими наборами даних та високою пропускною здатністю операцій. Загалом процес шардування відбувається за допомогою створення спеціального ключа сегментації.

Ключ сегментації – це індекс, розроблений на основі одного чи декількох об'єктів, призначений для створення умов розділення даних. Його не можна вибирати довільним чином, він має бути окремо підібраний до будь-якої системи, тобто не може мати універсального рішення. Ключ має відповідати вимогам, які, у свою чергу, допоможуть правильно його побудувати або вибрати.

Вибір ключа сегментування впливає на те, як балансувальник створює і розподіляє блоки за доступними шардами. Це змінює загальну ефективність і продуктивність усіх операцій усередині кластера. Також ключ сегментування впливає на продуктивність та ефективність стратегії сегментування, що використовується кластером.

"Ідеальний" ключ сегментування дозволяє MongoDB рівномірно розподіляти документи між шардами. Під цим поняттям слід розуміти такий ключ, що буде максимально задовольняти вимоги та повністю відповідати обмеженням ключа розподілення.

Обмеження ключа розподілення: його розмір не повинен перевищувати 512 байт; тип індексування має бути або простим зростаючим, або складеним, що включає в себе потрібні поля в порядку зростання їх значення, або хешованим; ключ повинен бути незмінним (не може бути можливості його динамічної зміни залежно від ситуації); незмінність ключа у колекції (не можна видалити його або якимось чином змінити для вже створеного розподілення); монотонне зростання ключа сегментації, що може призвести до обмеження пропускної здатності кластера.

Однією з найважливіших вимог до створення ключа є його потужність, яка визначає максимальну кількість блоків, що балансувальник може створити. Цей параметр може зменшити або звести нанівець ефективність горизонтального масштабування кластера. Унікальне значення ключа може існувати лише в одному шарді в будь-який момент часу. Наприклад, якщо ключ шарду має потужність, що дорівнює 4, то такий шард не може містити більше чотирьох блоків. Саме це число обмежує кількість ефективних сегментів у кластері, тому що додавання нових шардів не буде приносити жодної користі. На рис. 1 наведено розподілення даних у разі застосування ключа сегментування "x" низької потужності для тришардової системи.

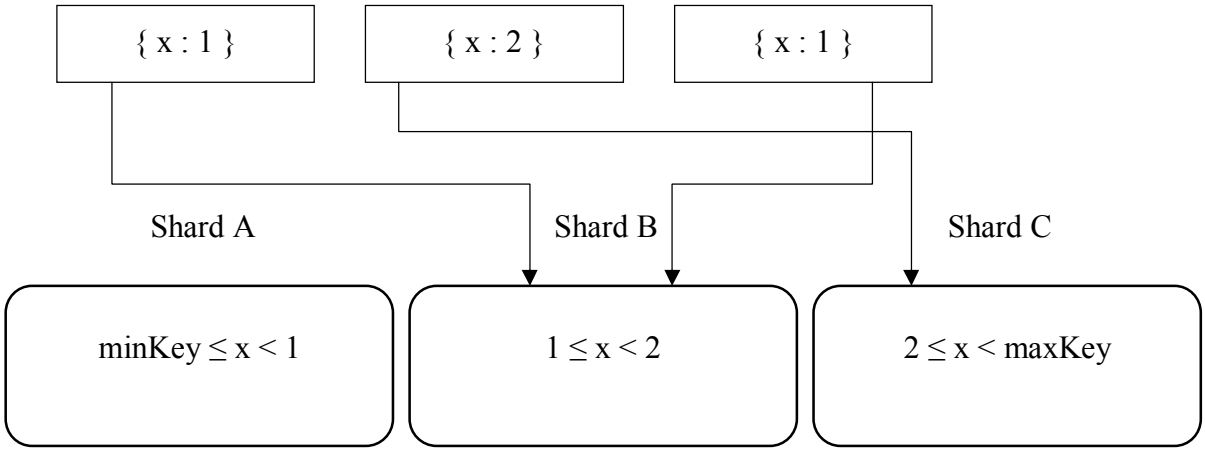

Рис. 1. Потужність ключа для тришардової системи

Необхідно зазначити, що ключ з великою потужністю не гарантує рівномірного розподілення даних за кластерами, хоча і полегшує горизонтальне масштабування. Частота ключа і швидкість його зміни (зростання) також впливає на розподілення даних, тому треба враховувати кожен фактор під час створення або вибору ключа. Якщо ж система вимагає використовувати ключ з низькою потужністю, то потрібно включити його до складеного індексу, у якому загальна потужність буде вищою.

Розглянемо набір даних, що становить спектр значень ключа: частота сегментів показує, як часто конкретне його значення зустрічається в даних. Якщо більшість документів містить лише набір цих значень, то такі блоки стануть вузьким місцем у кластері. Крім того, коли блоки зростатимуть, вони можуть стати нероздільними частинами, оскільки не можуть бути більше розділені. Усе це зменшує ефективність горизонтального масштабування.

На рис. 2 наведено розподілення даних у разі застосування ключа сегментування “x” великої частоти для тришардової системи.

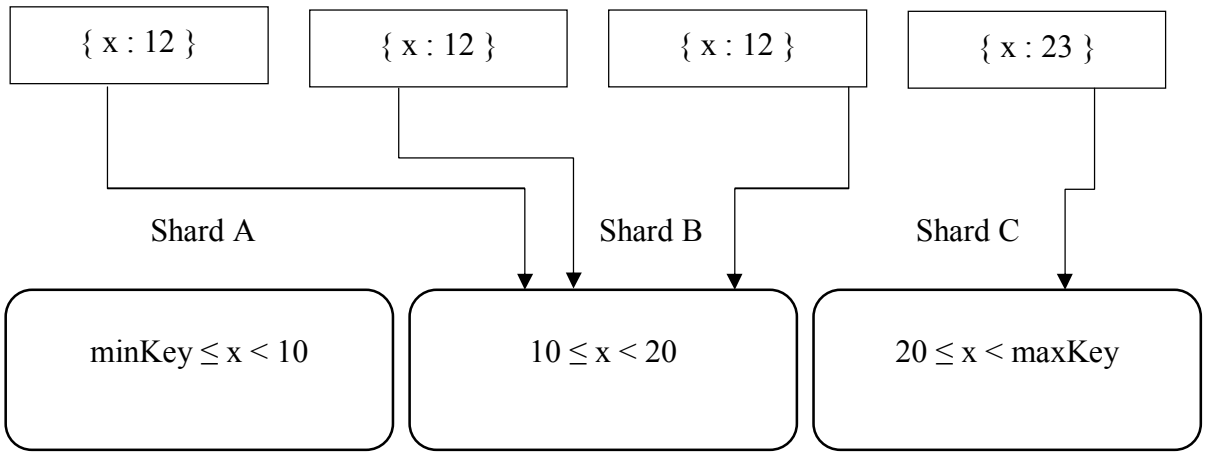

Рис. 2. Частота ключа

Ключ сегментації з низькою частотою не гарантує рівномірного розподілення даних за кластерами. Потужність і швидкість зміни основних сегментів також сприяє розподіленню даних за кластерами.

Якщо модель даних потребує сегментування на ключ, який має високу частоту, то рекомендується додавати його у складений індекс.

Ключ сегментування, який збільшується або зменшується монотонно, має великий шанс вставити дані в один шард замість розподілення інформації рівномірно. Це відбувається, якщо кожен кластер має блок, який фіксує діапазон даних з верхньою межею в змінну maxKey. Вона завжди порівнюється з усіма значеннями ключа та переприсвоюється. Аналогічно застосовується змінна з нижньою межею – minKey.

Якщо ключ шардування постійно зростає, то всі нові дані відправляються в блок з maxKey як верхню межу. І навпаки, якщо ключ постійно спадає, то такий шард стає вузьким місцем для операцій запису, це негативно впливає на систему в цілому.

На рис. 3 наведено розподілення даних у разі застосування монотонно зростаючого ключа сегментування "x" для тришардової системи.

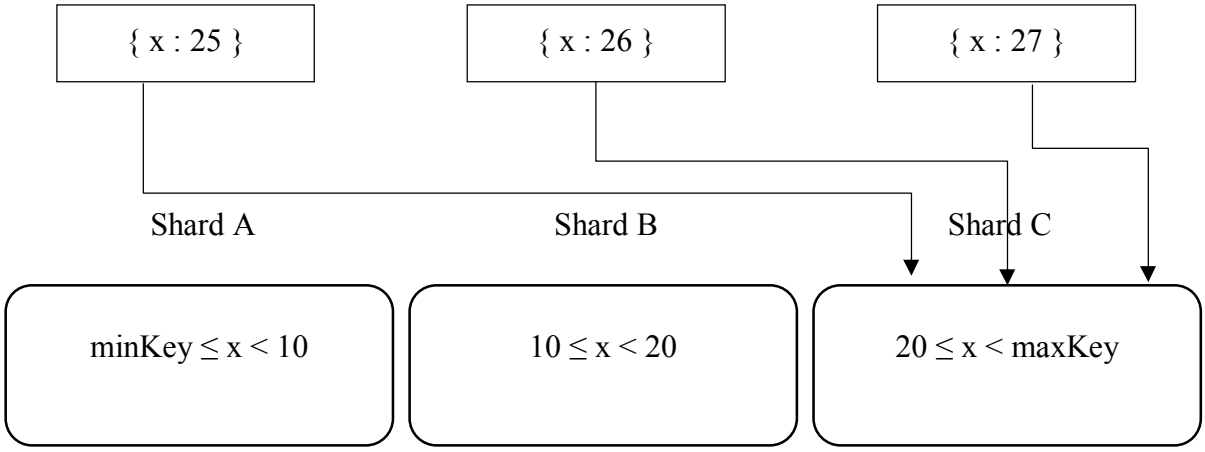

Рис. 3. Монотонне зростання ключа

Якщо ключ буде монотонно спадати, то вся інформація прямуватиме до шарда A і, навпаки, до шарда C під час зростання.

Ключ, який не змінюється монотонно, не дає гарантії рівномірного розподілу даних за кластерами. Потужність, частота та немонотонна зміна ключа – "три стовпи", на яких базується рівномірне розподілення даних між серверами. Треба враховувати кожен із факторів під час проектування бази даних та вибору ключа сегментації.

Якщо ж модель даних потребує сегментування через ключ, що монотонно змінюється, то рекомендується використовувати сегментування за допомогою ключа – хеша.

Для побудови моделі розподілення даних між шардами треба проаналізувати та систематизувати дані, що надходять до системи від користувачів, а також ті, з яких користувачі отримують потрібну інформацію. Базою дослідження обрано веб-сайт подачі безкоштовних оголошень, для якого можна виділити декілька особливостей даних для такого типу системи: динамічність, тобто вони змінюються постійно; неперіодичність, тобто немає чіткої математичної формули, що описує який об'єм даних отримає сервер у наступну хвилину/годину тощо.

Досить часто спостерігаються ситуації стрімкого зростання або зменшення кількості людей на сайті, а це, у свою чергу, відобразиться на швидкій чи повільній зміні даних.

Модель одного поданого оголошення складається з 33 полів, кожне з яких має свою функцію та несе корисну інформацію. Деякі з даних об'єктів є метаданими, що відіграють роль зв'язних аргументів з іншими таблицями бази даних (БД) – "зовнішніми ключами".

Для побудови моделі розподілення даних потрібно чітко виділити поля, які беруть участь у запитах користувачів щодо конкретної БД. Для отримання реальної картини запитів у системі, що функціонує, необхідно задіяти можливості підсистеми реєстрації подій.

Для отримання даних щодо реєстрації подій у реальній діючій системі було запропоновано використати чотирикрокову процедуру проксингу трафіка (рис. 4).

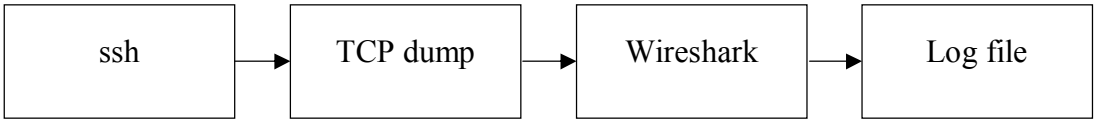

Рис. 4. Процедура проксингу трафіка

Дана процедура працює таким чином:

1. З консолі Linux засобами термінального додатка SSH підключаємося до сервера, на якому знаходиться основний кластер БД.

2. Через консольний додаток TCPDump збираємо всі звернення до порту служби MondoDB (27100) за протоколом TCP.

3. Отримані дані надходять до програмного аналізатора мережевого трафіка Wireshark.

4. У середовищі Wireshark здійснюється аналіз пакетів даних та розшифрування повідомлення протоколу MongoDB для отримання значущої інформації.

Для роботи з отриманими даними створено скрипт, що реалізує сокет типу SOCK_RAW для протоколу TCP. Даний сокет пропускає через себе всі пакеті TCP, що надходять до сервера, і відфільтровує їх за ознаками пакету Mongo, таким чином, отримаємо ті самі дані в байтовому вигляді. Для того, щоб виділити необхідну корисну інформацію, створено шаблон аналізу структури пакету типу OP_COMMAND, який використовується системою запитів від користувачів. Другою функцією скрипта є розбір пакету за шаблоном та створення лог-файлу, зручного для нас вигляду, для накопиченої за час дослідження статистики.

Після збору запитів користувачів та трансформації їх у зручну форму розроблено алгоритм для побудови статистики, його суть полягає в такому:

1. Отримання зі сховища усіх очищених до потрібної (JSON) форми запитів.

2. Розкладання JSON-об'єкта на асоціативний масив даних.

3. Виконання (за допомогою рекурсивної функції) обходу до максимального рівня вглибину асоціативного масиву даних.

4. До кожного поля (ключа), що зустрічається в запиті під час проходження через функцію, додавання + 1.

5. Отримання розподілу частоти ключів за запитами користувачів.

Під час збирання статистики проаналізовано 1055148 запитів за період з 5 по 26 липня 2017 року. Результати наведено на рис. 5.

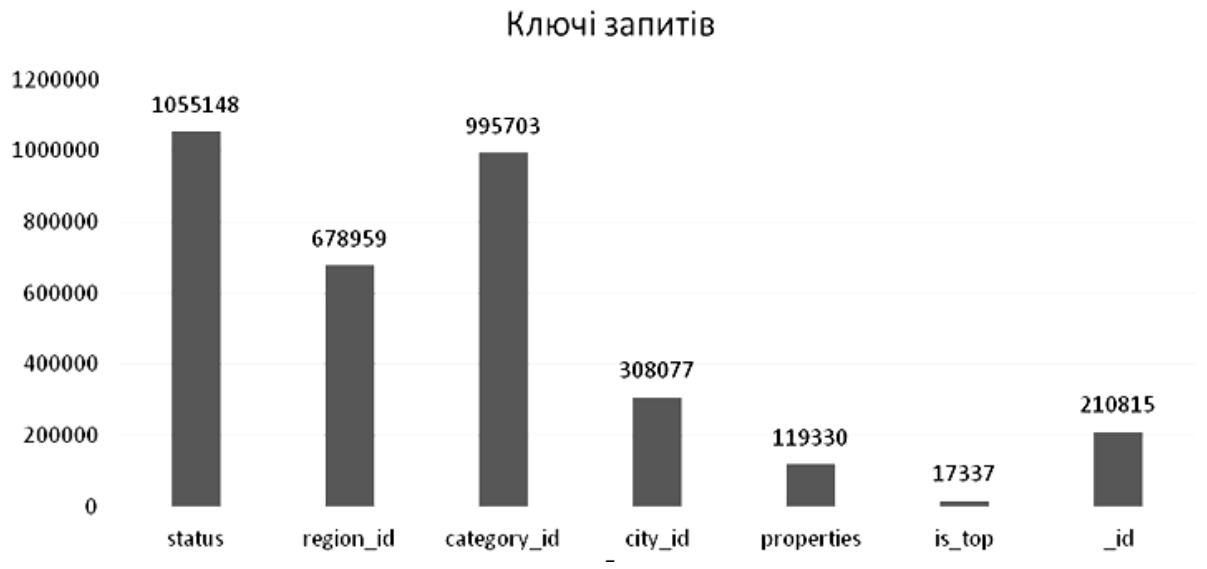

Рис. 5. Розподілення ключів запитів користувачів

Після виведення результатів для простоти їх оцінювання все зведено до процентного відношення участі ключа в запитах. У підсумку отримуємо такий результат: поле *"status"* бере участь в 100% запитів, адже всі оголошення на сайті розподіляються за статусом (деякі доступні всім користувачам, інші знаходяться на модерації, а окремі взагалі видалені із сайту і позначаються спеціальним статусом у БД); поле *"category_id"* використовується в 94% участі в запитах. Даний ключ розподіляє всі оголошення за категоріями, тому логічно визначити його участь у майже всіх запитах на сайті; поле *"region_id"* використовується в 64%, це ідентифікатор належності оголошення до певного регіону України (кожне з оголошень має бути територіально прив'язане згідно з політикою правил сайту; зазвичай люди шукають оголошення в певному регіоні, де їм було б зручніше перевірити якість товару або послуги); поле "*city_id*" використовується в 29% запитів, аналогічно до регіону користувачі уточнюють місцеположення товару чи послуги в містах, але даний ключ не користується такою популярністю, оскільки користувачам головне, у якому регіоні знаходиться оголошення, тому що реально за допустимий час доїхати до будь-якого міста, яке знаходиться в тому самому регіоні, що й користувач; поле *"properties"* використовується в 11%, воно містить властивості оголошень (розмір, вага, пробіг, кількість кімнат тощо); поле *"is_top"* використовується в 2%, воно містить властивість платної функції "топ-оголошення"; поле *"_id"* застосовується в 20% запитів оскільки містить ідентифікатор оголошення, зазвичай використовується для пошуку пропозиції за допомогою спеціального пошукового механізму Sphinx та подальшої передачі ідентифікаторів знайдених об'єктів до MongoDB.

Статистика показала багато параметрів, що беруть участь у вибірках з БД, але їх відношення до загального об'єму запитів дуже мале (менше 1%). Оцінюючи вплив таких параметрів на загальну роздільну здатність як дуже малий, ними можна знехтувати.

Для побудови ключа розподілення можна було б взяти всі згадані вище ключі та, спираючись на них, розробляти згадану модель. Якщо ж прийняти до уваги кількість значень певного ключа (табл. 1), то деякі з них варто відкинути. У першу чергу потрібно знехтувати тими, що мають замалу або завелику кількість значень, бо вони заважатимуть формуванню ефективного ключа.

Таблиця 1

Розподіл ключів у БД

Назва ключа	Діапазон значень	Кількість значень	Відношення
Status	[1, 15]	7	100%
Category_id	[1, 1755]	1010	94%
Region_id	[1, 25]	25	64%
City_id	[1, 720]	677	29%
Properties	[1, 122600]	122000	11%
Is_top	[0,1]	2	2%
_id	[6145245, 21611833]	4651647	20%

Наявні поля “properties” та “is_top” відкидаємо з таких причин: “properties” має дуже великий діапазон для незначної кількості входжень у запити; а “is_top” вирізняється малим відношенням порівняно з іншими ключами. Тому для побудови моделі залишається набір таких полів: “status”, “category_id”, “region_id”, “city_id”, “_id”.

Для побудови оптимальної моделі розподілу було прийнято рішення поєднати обидві моделі даних для отримання максимальної оптимальності роботи системи розподілу (рис. 6).

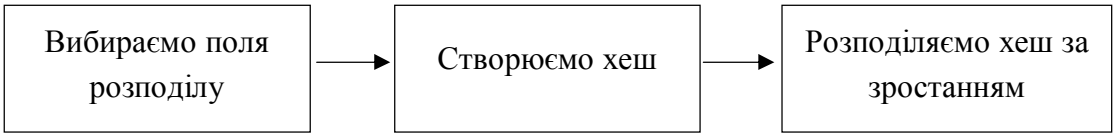

Рис 6. Модель розподілу

Для побудови моделі хеша скористаємося відомими даними з табл. 1. Результат наведено на рис. 7.

S	04	ct	01302	r	013	c	303	_id	011648102

Рис. 7. Модель хеша

Розглянемо модель хеша детальніше: s – статус; 04 – значення статусу; ct – категорія; 01302 – ідентифікатор категорії 01302; r – регіон; 013 – ідентифікатор регіону; С – місто; 303 – ідентифікатор міста; _id – ключ, ідентифікатор оголошення; 011648102 – значення ідентифікатора оголошення.

Порядок розташування елементів у хеші відповідає відношенню значущості ключів у запитах користувачів, крім ідентифікатора оголошення, який поставлено в кінець для створення унікальності хеша.

Для визначення ефективності роботи моделі розподілення було проведено три незалежні тести з різними моделями. Для тестування було створено БД “з нуля”, у ній

не виконували ніяких запитів. Було проведено по три тести кожної моделі. Перед кожним проходженням очищували хеш запитів на усіх шардах БД для максимальної чистоти експерименту.

Для проходження тестування ефективності було вибрано 80 тисяч запитів, однакових для кожної ітерації. У тесті брали участь такі моделі:

а) без розподілення даних за шардами (табл. 2);

б) стандартне розподілення за основним ключем – id оголошення (табл. 3);

в) розподілення за створеним ключем згідно з розробленою моделлю (табл. 4).

Результати тестування для зазначених моделей наведено в табл. 2–4 відповідно.

Таблиця 2

№ з/п	Виконання тесту (у секундах)	Завантаження CPU / RAM
1.	673.74932098389	100% CPU, 49% RAM
2.	362.3379099369	100% CPU, 49% RAM
3.	361.94748401642	100% CPU, 49% RAM

Таблиця 3

№ з/п	Виконання тесту (у секундах)	Завантаження CPU / RAM
1.	298.05477404594	100% CPU, 35% RAM
2.	194.26065301895	45-50% CPU, 35% RAM
3.	193.08583283424	45-50% CPU, 35% RAM

Таблиця 4

№ з/п	Виконання тесту (у секундах)	Завантаження CPU / RAM
1.	298.94256091118	100% CPU, 22.9% RAM
2.	60.710704088211	40-45% CPU, 22.9% RAM
3.	60.328711032867	40-45% CPU, 22.9% RAM

Після отримання результатів тестування, можна зробити висновки: розроблена модель показала зростання швидкості читання даних з 2-ї ітерації **втричі**; вдалося зменшити рівень користування оперативною пам'яттю порівняно зі звичайним розподіленням за "_id"; просте розподілення даних збільшує швидкість читання даних **удвічі**.

Перший тест у двох моделях з шардування показав однаковий час, це пов'язано з наповненням пам'яті індексними полями в MongoDB, тобто для кращого перезапуску служби на продакшн-сервері можна "прогріти" MongoDB за допомогою виконання обмеженої кількості запитів, тим самим прискорюючи їх виконання у звичайних користувачів. Оскільки id оголошення унікальний, то і хеш до кожної моделі також унікальній.

Таке слідування в хеші забезпечує спеціальний порядок розподілу оголошень за шардами, тобто ми застосовуємо модель хешування з моделлю ранжування й отримуємо дуже цікаву процедуру розміщення об'єктів оголошень. Наприклад, коли користувач шукає оголошення за категорією (відомо, що 100% робить запит з яким-небудь статусом),

імовірніше за все, що інформація знаходиться на одному сервері, тому це пришвидшує отримання даних.

Висновки. У статті запропоновано комбіновану модель диспетчеризації даних між кластерами та новий метод їх розподілення, що відрізняється від відомих підходом до розгляду ключа розподілення даних. Крім того, модифіковано та досліджено моделі розподілення даних у базі документоорієнтованого типу MongoDB із використанням методів оптимального розподілення даних. Розроблена модель показала зростання швидкості читання даних з 2-ї ітерації втричі, також вдалося зменшити рівень користування оперативною пам'яттю.

СПИСОК ЛІТЕРАТУРИ

1. Baker J. et al. Megastore: Providing Scalable, Highly Available Storage for Interactive Services // Proc. 5th Biennial Conf. Innovative Data Systems Research (CIDR 11). ACM, 2011. P. 223-234.
2. Brewer E. Lessons from Giant-Scale Services // IEEE Internet Computing, July–Aug., 2001. P. 46-55.
3. Mahajan P., Alvisi L., Dahlin M. Consistency, Availability and Convergence, tech. report UTCS TR-11-22. Univ. of Texas at Austin, 2011. 31 p.
4. Rick C. MongoDB Applied Design Patterns: Practical Use Cases with the Leading NoSQL Database. Copeland Rick., 2013. 176 p.
5. Kristina Chodorow. Scaling MongoDB: Sharding, Cluster Setup and Administration. O'Reilly, 2011. 66 p.
6. Optimization Strategies for MongoDB. URL: https://docs.mongodb.com/v3.0/administration/optimization/.

Подано 13.07.2018

А. А. Єфименко, В. Н. Ковальчук, Г. О. Мишин, И. И. Сугоняк

МОДЕЛЬ ДИСПЕТЧЕРИЗАЦИИ ПОТОКОВ ДАННЫХ ДЛЯ ВЫСОКОНАГРУЖЕННЫХ ВЕБ-СИСТЕМ

В статье предложена модель оптимального распределения данных между кластерами (серверами) в условиях высоконагруженной системы и описана построенная на их основе прикладная веб-ориентированная система. Как промежуточный сервер использована нереляционная система управления базами данных MongoDB, для которой реализован алгоритм сбора статистики и построения ключа распределения данных. Для получения информации о регистрации событий в реальной действующей системе было предложено использовать чотрехшаговую процедуру проксинга трафика. После сбора запросов пользователей и трансформации их в удобную форму разработан алгоритм для построения статистики. Было проведено три независимых теста с такими моделями: без распределения данных по шардам; стандартное распределения по основному ключу; распределения по созданному ключу согласно разработанной модели. Выбрано 80000 запросов, одинаковых для каждой итерации. Эффективность шардинга данных на основе ключа распределения подтверждена тестовыми экспериментами.

Ключевые слова: *оптимизация запросов, шардинг, высоконагруженные системы, SQL Server, MongoDB.*

A. A. Yefimenko, V. N. Kovalchuk, G. O. Mishin, I. I. Sugonyak

HIGHLY LOADED WEB SYSTEM'S MODEL FOR STREAMING DATA FLOWS

The model for optimal data allocation between clusters (servers) in a highly loaded system and built-in application web-based system have been proposed in paper., the non-relational DBMS MongoDB is used as an intermediate server, the algorithm for collecting statistics and constructing a data split key is implemented. It was proposed a four-step procedure using for traffic proxies to data ordering on event logging. The lgorithm for constructing statistics is developed, when user queries was collecting and transforming into convenient form. There were 3 independent tests with the following models: without the distribution of data for shards; standard distribution by main key; distribution for the created key according to the developed model. 80,000 requests are selected for each iteration. The efficiency of data shearing based on the distribution key is confirmed by test experiments.

Keywords: *query optimization, shading, high-capacity systems, SQL Server, MongoDB.*