5. *David Bihanic. New Challenges for Data Design // 12.3. Case Study: The Deleted City Springer, 2014. — 447 p. — P. 223—232.*
6. *Сполучені Штати. "GeoCities залучає майже 180 мільйонів відвідувачів в Інтернеті щороку". Siteanalytics.compete.com.Архів від Оригінальний 20 грудня 2011 року. Отримано 20 лютого, 2012.Енциклопедія site:uk.wikisko.ru*
7. *Рао, Ліна (23 квітня 2009 р.). "Yahoo тихо втягує штепсель у GeoCities". TechCrunch. В архіві від оригіналу 2 червня 2009 р. Отримано 30 квітня, 2009.Енциклопедія site:uk.wikisko.ru*
8. *"FTC.gov" (PDF URL: https://web.archive.org/web/20090511204518/http://www.ftc.gov/os/decisions/docs/Volume127.pdf). від Оригінальний (PDF URL: https://www.ftc.gov/os/decisions/docs/Volume127.pdf#page=94) 11 травня 2009 р.*

УДК 004.62

Солодун Т. Р., студентка
Зелінська О. В., к.т.н., доцент, доцент кафедри інформаційних технологій

ПОМИЛКИ В СИСТЕМАХ БАЗ ДАНИХ І ТЕОРЕМА CAP

Донецький національний університет імені Василя Стуса, м. Вінниця

Анотація. У даному дослідженні подана інформація про помилки в системах баз даних, розглянуто теорему CAP. Методологічною основою роботи є розгляд теореми CAP та її відношення до СУБД. Специфіка досліджуваної теми передбачає розгляд даного поняття на прикладі використання у СУБД.

Ключові слова: CAP, транзакції, помилковість, БД, СУБД, NoSQL.

Усі коли-небудь бачили рекламу, яка починається із заголовка «Дешево, швидко і добре: виберіть два». Теорема CAP застосовує подібний тип логіки до розподілених систем, а саме, що розподілена система може забезпечувати лише дві з трьох бажаних характеристик: узгодженість, доступність і толерантність до розділів ("C", "A" і "P" в CAP).

Розподілена система — це мережа, яка зберігає дані на більш, ніж одному вузлі (фізичних або віртуальних машинах) одночасно. Оскільки всі хмарні програми є розподіленими системами, важливо розуміти теорему CAP під час розробки хмарного додатка, щоб ви могли вибрати систему керування даними, яка забезпечує характеристики, які найбільше потребують вашій програмі.

Теорему CAP також називають теоремою Брюера, оскільки вона була вперше висунута професором Еріком А. Брюером під час доповіді про розподілені обчислення у 2000 році. Два роки потому професори

Массачусетського технологічного інституту Сет Гілберт і Ненсі Лінч опублікували доказ «гіпотеки Брюера».

Загалом, теорема стверджує, що є три властивості, які вимагаються для додатків СУБД:

C: (Consistency) Послідовність. Мета — дозволити багатосайтовим транзакціям мати звичну семантику «все або нічого», яка зазвичай підтримується комерційними СУБД. Крім того, коли копії підтримуються, потрібно, щоб копії завжди мали узгоджені стани.

A: (Availability) Доступність. Метою є підтримка СУБД, яка завжди працює. Іншими словами, коли відбувається збій, система повинна продовжувати роботу, перемикаючись на копію, якщо потрібно. Ця функція була популяризована Tandem Computers більше 20 років тому.

P: (Partition-tolerance) Толерантність до розділу. Якщо відбувається збій мережі, який розділяє вузли обробки на дві групи, які не можуть спілкуватися один з одним, тоді мета полягатиме в тому, щоб обробка продовжувалася в обох підгрупах.

У спільноті NoSQL ця теорема була використана як виправдання відмови від узгодженості. Оскільки більшість систем NoSQL зазвичай забороняють транзакції, які перетинають межі вузла, то узгодженість застосовується лише до реплік. Тому теорема CAP використовується для виправдання відмови від послідовних реплік, замінюючи цю мету на «можливу узгодженість». З цим розслабленим поняттям можна лише гарантувати, що всі репліки в кінцевому підсумку зберуться в один і той же стан, тобто коли мережеве з’єднання буде відновлено і пройшло достатньо часу для очищення репліки. Виправдання відмови від C полягає в тому, щоб можна було зберегти A і P. Детальніше показано додатки, що підтримують відмову від одного з пунктів у Рисунок 1 [3].

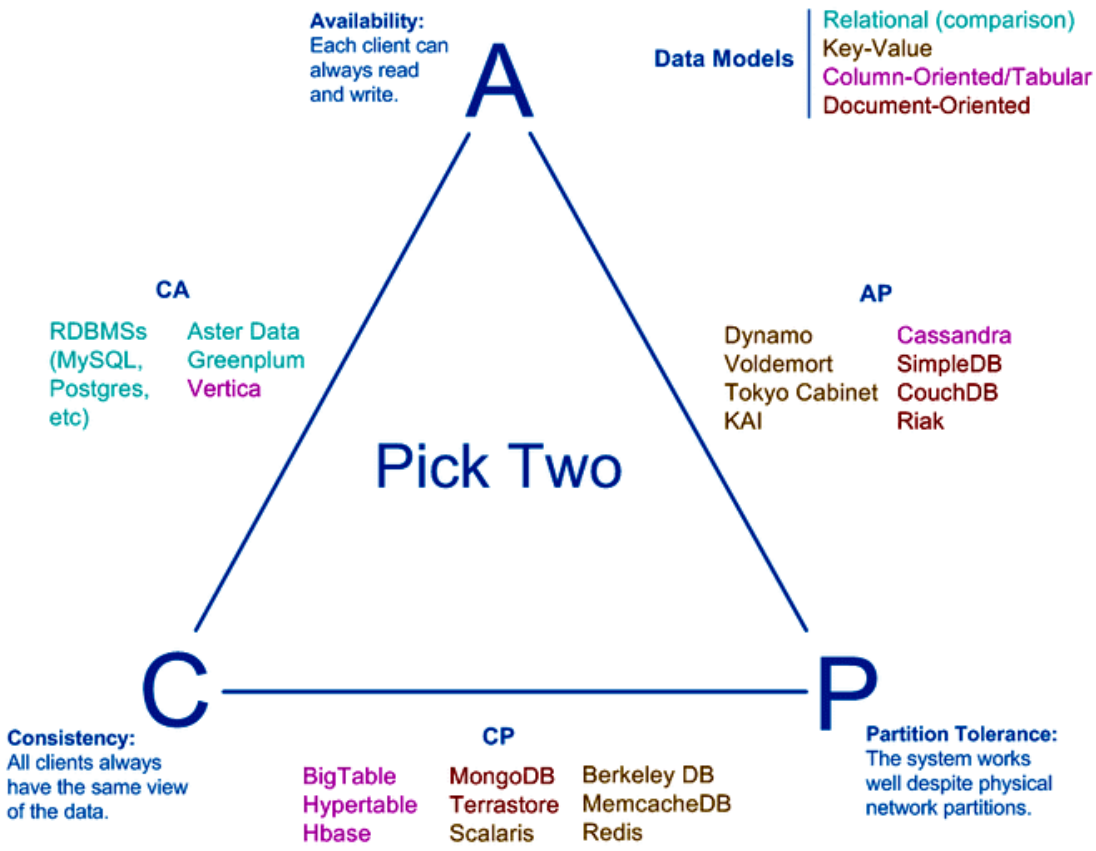

Рисунок 1

Однак відновлення після помилок має більше аспектів, які слід розглянути. Ми припускаємо типову апаратну модель набору локальних вузлів обробки та зберігання, зібраних у кластер за допомогою мережі локальної мережі. Кластери, у свою чергу, об’єднані разом за допомогою мережі WAN.

Давайте почнемо з обговорення того, що викликає помилки в базах даних. Нижче наведено принаймні частковий список:

1) Помилки програми. Програма виконала одне або кілька неправильних оновлень. Як правило, це не виявляється протягом кількох хвилин або годин після цього. Необхідно створити резервну копію бази даних до певної точки перед транзакцією-порушником, а подальшу діяльність повторити.

2) Повторювані помилки СУБД. Збій СУБД на вузлі обробки. Виконання тієї ж транзакції на вузлі обробки з копією приведе до збою резервної копії. Ці помилки були названі помилками Бора.

3) Неповторні помилки СУБД. База даних зламалася, але репліка, ймовірно, буде в порядку. Вони часто викликані дивними випадками, пов’язаними з асинхронними операціями, і їх називають Heisenbugs [2]

4) Помилки операційної системи. Операційна система зірвалася на вузлі, викликаючи «синій екран смерті».

5) Збій обладнання в локальному кластері. Сюди входять збої пам’яті, збої диска тощо. Зазвичай вони викликають «панічну зупинку» ОС або СУБД. Однак іноді ці збої проявляються як Heisenbugs також.

6) Розділ мережі в локальному кластері. Помилка локальної мережі, вузли більше не можуть спілкуватися один з одним.

7) Катастрофа. Відбувається рідкісна незалежна подія (наприклад, повінь), тому що зниження продуктивності для її уникнення занадто високе

8) Збій мережі в WAN, що з'єднує кластери разом. Збій глобальної мережі, і кластери більше не можуть спілкуватися один з одним.

Варто звернути увагу, що помилки 1 і 2 спричинять проблеми з будь-якою схемою високодоступності. У цих двох сценаріях немає можливості продовжувати; тобто доступності неможливо досягти. Крім того, узгодженість реплік не є найкращим рішенням; поточний стан СУБД уже неправильний. Помилку 7 можна буде відновити лише в тому випадку, якщо локальна транзакція буде здійснена лише після впевненості, що транзакція була отримана іншим кластером, підключеним до глобальної мережі. Мало хто з розробників програм готові прийняти таку затримку.

Отже, остаточну узгодженість не можна гарантувати, оскільки транзакція може бути повністю втрачена, якщо катастрофа станеться в локальному кластері до того, як транзакція буде успішно переслана в інше місце. Розробник вирішує зазнати втрати даних у даній ситуації.

Таким чином, помилки 1, 2 і 7 є прикладами випадків, для яких теорема CAP просто не застосовується. Будь-яка реальна система повинна бути готова мати справу з відновленням у цих випадках.

Тепер перейдемо до випадків, коли може застосовуватися теорема CAP. Враховуючи локальні збої (3, 4, 5 і 6), переважна більшість спричиняє збій одного вузла, що виходить з ладу, - випадком мережевого розділу, який легко виживає за допомогою багатьох алгоритмів. Тому, на мою думку, набагато краще відмовитися від P, ніж пожертвувати C. (У середовищі локальної мережі, я думаю, слід вибрати CA, а не AP). Новіші системи SQL OLTP (наприклад, VoltDB і NimbusDB), роблять саме це.

Розглянемо помилку 8, розділ у мережі WAN. У сучасних глобальних мережах є достатньо резервування, що розділи зустрічаються досить рідко. Локальні збої та помилки програми є набагато більш імовірними. Більше того, найімовірніший збій WAN – це відокремлення невеликої частини мережі від більшості. У цьому випадку більшість може продовжувати роботу з простими алгоритмами, і лише мала частина повинна блокувати. Отже, здається нерозумним постійно відмовлятися від узгодженості в обмін на доступність невеликої підмножини вузлів у досить рідкісному сценарії.

Нарешті, розглянемо уповільнення роботи ОС, СУБД або менеджера мережі. Це може бути викликано перенавантаження, проблемами з буферним пулом або незліченною кількістю інших причин. Єдине рішення, яке можна прийняти в цих сценаріях, — це «провалити» порушений компонент; тобто перетворити повільний час відповіді на невдачу в одному із випадків, згаданих раніше. Можна просто перенести проблему в інше місце і додати помітне навантаження на обробку для подальшого відновлення. Крім того, подібні проблеми завжди виникають при великому навантаженні – вирішення цього шляхом віднімання обладнання йде в неправильному напрямку.

Очевидно, що потрібно писати програмне забезпечення, яке без збоїв може впоратися зі стрибками навантаження; наприклад, при зниженні навантаження

або роботі в зниженому режимі. Крім того, якісне програмне забезпечення для моніторингу допоможе виявити такі проблеми на ранній стадії, оскільки реальним рішенням є збільшення потужності. Нарешті, безумовно, непоганою ідеєю є програмне забезпечення, яке самостійно змінює налаштування, яке може швидко поглинати додаткові ресурси.

Отже, не слід викидати C так швидко, оскільки існують реальні сценарії помилок, коли CAP не застосовується, і це здається поганим компромісом у багатьох інших ситуаціях.

Список літератури

1. https://www.ibm.com/cloud/learn/cap-theorem#toc-what-is-th-DXABpEgu

2. Jim Gray, "Why Do Computers Stop and What Can be Done About It," Tandem Computers Technical Report 85.7, Cupertino, Ca., 1985. http://www.hpl.hp.com/techreports/tandem/TR-85.7.pdf

3. https://habr.com/ru/post/328792/