ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ ВИРІШЕННЯ Й ЗАПОБІГАННЯ ВЗАЄМОБЛОКУВАННЯ

Ковальова Т.В., Андрюхін О.І.
Донецький національний технічний університет

У цій доповіді розглядаються алгоритми вирішення й запобігання взаємоблокування, причини появи взаємоблокувань у різних СУБД. Основна мета - визначити найбільш ефективні методи боротьби із взаємоблокуванням, дослідити методи вирішення й запобігання взаємоблокування у різних СУБД.

При паралельному виконанні процесів можуть виникати такі ситуації, при яких два й більше процеси увесь час перебувають у стані блокування. Про такі процеси говорять, що вони перебувають у стані взаємного блокування, тупика, дедлока (deadlock) або клінчу (clinch) ("смертельних обіймів"- deadly embrace). Із проблемою тупиків тісно пов'язана проблема нескінченного відкладання, коли процес, що навіть не перебуває в стані тупика, очікує дії, що може ніколи не відбутися через "необ'єктивні" принципи, закладені у системі планування ресурсів. Одним з найпростіших прикладів тупика при розподілі ресурсів є випадок, коли кожний із двох процесів чекає ресурс, зайнятий іншим процесом; через це очікування жоден із процесів не може продовжити виконання й звільнити ресурс, необхідний іншому [1].

Відповідно до прийнятої класифікації алгоритми визначення й вирішення deadlock можна умовно розділити на три групи: алгоритми централізованого, розподіленого й ієрархічного керування [2].

Для централізованих алгоритмів необхідний глобальний граф очікувань всієї мережі, що є практично неможливим у цей час через розмір реальні мережі. Розподілені алгоритми через їхнє практичне значення є найбільш досліджувані. Виділяють чотири типи [3].

1.Алгоритми проштовхування шляху (Path-pushing), у яких передається інформація про необхідний шлях просування від вузлів очікування до блокованих вузлів.

2.Алгоритми прогону (Edge-chasing), у яких визначення циклів ініціюється вузлом, що не може виконувати операції через блокування своїх дій й який генерує пробні повідомлення. Ці повідомлення змінюють характеристики заблокованих вузлів при своєму просуванні.

3.Дифузійні алгоритми, у яких ініціація дифузійних обчислень визначення циклів виконуються незаблокованими процесами й використовуються більше складні моделі deadlock, ніж в алгоритмах прогону

4.Алгоритми визначення глобального стану розподіленої мережі. У них виконується спроба одержання дампа глобального WFG.

В ієрархічних алгоритмах процеси групуються в незалежні кластери. Періодично визначається центральний контрольний кластер, що динамічно вибирає керуючий центр для кожного кластера

Як правило, у СУБД присутні механізми визначення подібних тупикових ситуацій й їхнього усунення. Розглянемо такі способи [3]:

1. Timeout based

Найпростіший спосіб - це ввести деякий фіксований час очікування (timeout), і якщо транзакція виявилася заблокованою більше цього часу, то вважати, що вона ввійшла в тупикову ситуацію й скасувати її. Недоліки цього способу очевидні - немає

стовідсоткової гарантії, що скасована транзакція була однією з учасниць взаємоблокування. Збільшення часу тайм-ауту підвищує цю ймовірність, але одночасно збільшує час виявлення дійсно намертво заблокованих транзакцій. А це, у свою чергу, веде до збільшення часу простою запитів, що не беруть участь у взаємоблокуванні і ресурсів, що очікують, захоплених намертво заблокованими транзакціями.

2. Wait-for graph based

Менеджер блокувань будує спрямований граф, що називається «графом очікування» (wait-for graph). У вершинах цього графа перебувають транзакції, а в ребрах – залежності. Наприклад, ребро T_i->T_j з'являється в тому випадку, якщо T_i чекає, доки T_j об'єкт. Таким чином, якщо в графі очікування виникає цикл (T_1->T_2->…->T_n->T_1), то T_1 чекає сама себе, як і всі інші n транзакцій у циклі, отже, транзакції заблоковані намертво. У цьому випадку виявлення взаємоблокування - це знаходження замкнутих циклів у графі очікування. Самі залежності в граф додаються й знищуються в міру одержання й зняття блокувань. Складність полягає в тім, як часто менеджер блокувань повинен перевіряти граф очікування на наявність циклів. Теоретично це можна робити щораз при додаванні нової залежності, однак робити перевірки так часто нераціонально, оскільки, як правило, кількість звичайних блокувань набагато вище мертвих. Тому перевіряти наявність циклів можна або коли в граф додається якась фіксована кількість граней, або після закінчення якогось таймаута. Але тут, на відміну від попереднього способу, гарантується, що буде знайдена саме мертве блокування, а також, що ми знайдемо всі мертві блокування, а не тільки ті, які протрималися досить довго.

3. Timestamp based

Існують механізми, що дозволяють взагалі не допускати тупикових ситуацій при використанні протоколу двофазного блокування, наприклад, на основі тимчасових міток (timestamp).

Принцип, покладений в основу цього способу, досить простий. Кожної транзакції присвоюється тимчасова мітка. Далі можливі події:

1. «очікування-загибель» (wait-die). Якщо транзакція T_1 «старше» T_2, тоді транзакції T_1 дозволяється перебувати в стані очікування на блокуванні.
2. «поранення-очікування» (wound-wait). Якщо транзакція T_1 «старше» T_2, тоді T_1 «ранить» T_2 – транзакція T_2 відкочується, якщо тільки до моменту одержання «поранення» T_2 не виявляється вже завершеної. У цьому випадку T_2 «виживає» і відкоту не відбувається. Якщо ж T_1 «молодше» T_2, тоді T_1 дозволяється перебувати в стані очікування на блокуванні.

Цей спосіб простіший в реалізації, ніж побудова й відстеження графа очікування, відсутня ймовірність циклічного рестарту скасованої транзакції, тому що при скасуванні тимчасова мітка зберігається, і будь-яка транзакція згодом гарантовано стане самою «старшою», а тому її не скасують.

Література

[1] Гарви М. Дейтел Введение в операционные системы М: Мир, 1987

[2] E.Knapp. Deadlock detection in distributed databases.ACM Computing Surveys, 19(4):303–328, December 1987.

[3] A.D. Kshemkalyani and M. Singhal. Distributed detection of generalized deadlocks. In Proc. of the 17th Intl.Conf. on Distributed Computing System, pages 553–560, 1997.