УДК 004.72,004.738

Черкасов Д. І.

ПРИЧИННО-НАСЛІДКОВЕ ВПОРЯДКУВАННЯ ПОДІЙ НА ВУЗЛАХ ТЕРИТОРІАЛЬНО РОЗПОДІЛЕНОЇ СИСТЕМИ

Для забезпечення високої масштабованості, відмовостійкості, а також для впровадження сервісів на великих географічних територіях інформаційні системи будуються як сукупність розподілених вузлів, розташованих таким чином, щоб оптимізувати інформаційні потоки та максимально наблизити точку надання сервісу до споживача. Однією з найважливіших задач, яка розв'язується під час побудови розподілених систем, є здійснення синхронізації між вузлами системи, тобто приведення їх у взаємоузгоджений стан та реплікація даних між ними. Задача синхронізації може розглядатися як обмін повідомленнями, внаслідок якого кожен із вузлів потрапляє в потрібний стан. Унаслідок затримок передачі повідомлень між вузлами, які рознесені на значну відстань, можлива втрата причинно-наслідкового порядку створення повідомлень. У той час як відтворення правильного хронологічного порядку може досягатися за допомогою часових відміток повідомлень, факти появи певних повідомлень як реакції на певні події можуть бути втрачені. Для збереження під час синхронізації вузлів розподіленої системи не лише хронологічного, а й логічного порядку подій потрібно застосовувати причинно-наслідкове їх упорядкування.

Ключові слова: синхронізація, розподілена система, синхронна реплікація, асинхронна реплікація, причинно-наслідкове впорядкування.

Обмін повідомленнями як метод синхронізації вузлів розподіленої системи

Покриття мережею Інтернет усієї планети зумовило появу територіально розподілених систем, які складаються з вузлів, рознесених на значну відстань один від одного. З одного боку, такі системи здатні зменшити латентність під час обслуговування користувачів, оскільки кожен користувач обслуговується найбільш наближеним до нього вузлом, але, з іншого боку, виникає проблема синхронізації вузлів, тобто приведення їх до єдиного стану з погляду логіки функціювання системи. Синхронізація здійснюється за допомогою повідомлень, які надсилаються між вузлами. Вузли, на яких відбуваються певні події, що змінюють їхній стан, надсилають повідомлення іншим вузлам, спонукаючи їх також до відповідної зміни стану. У разі узгодженого і своєчасного обміну повідомленнями всі вузли системи перебувають в однаковому стані.

Як приклад розглянемо розподілену систему обміну повідомленнями між користувачами великої інтернет-спільноти. Кожен із користувачів надсилає свої повідомлення спільноті і отримує повідомлення від інших користувачів. Синхронізація вузлів системи обміну повідомленнями повинна забезпечити ідентичний склад та порядок відображення повідомлень усіх користувачів на всіх вузлах. Формування правильної послідовності повідомлень усіх користувачів не становить проблеми, якщо вузли системи наближені один до одного і передача повідомлень між ними відбувається майже миттєво – затримкою передачі повідомлення з вузла на вузол можна знехтувати. У ситуації рознесення вузлів на значну відстань затримка передачі повідомлення може стати причиною ускладнення синхронізації вузлів.

Розглянемо систему з двох вузлів – A та B (рис. 1). На вузлі A відбувається подія, яка змінює його стан. Для синхронізації – спонукання вузла B до аналогічної зміни стану вузол A надсилає вузлу B повідомлення MSG_A. Повідомлення надійде до вузла B із затримкою T_AB, після чого вузли будуть синхронізовані. У разі, якщо повідомлення передаються лише в одному напрямку (наприклад, від вузла A до вузла B) або часовий інтервал між подіями, що відбуваються на обох вузлах, значно перевищує затримку передачі, з синхронізацією не виникає проблем.

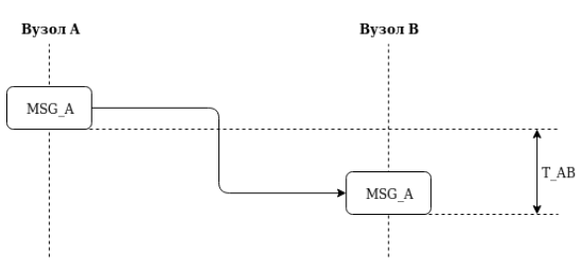

Рис. 1. Передача повідомлення між територіально рознесеними вузлами

Синхронізація ускладнюється, якщо часовий інтервал між подіями на вузлах є меншим, ніж затримка передачі. Розглянемо ситуацію, коли вузол B надсилає повідомлення MSG_B до вузла A пізніше, ніж вузол A відправив повідомлення MSG_A, але раніше, ніж MSG_A було отримано вузлом B (рис. 2). Порядок надходження повідомлень на вузлах буде різним.

Для формального запису порядку подій у системі скористаємося синтаксисом мови програмування Python для списків (lists). На вузлі A порядок подій виглядатиме так (перший елемент списку NONE додано для позначення початкового стану системи, в якому вона перебувала до передачі першого повідомлення):

```
Order_A = [ NONE, MSG_A, MSG_B ]
```

Натомість на вузлі B порядок подій виглядатиме так:

```
Order_B = [ NONE, MSG_B, MSG_A ]
```

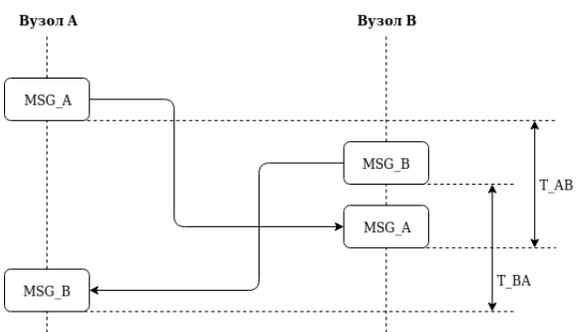

Рис. 2. Порушення порядку надходження повідомлень на територіально рознесених вузлах

Для забезпечення ефективної синхронізації, зокрема для відтворення однакового порядку надходження повідомлень на всіх вузлах, можливе використання таких підходів.

Синхронна реплікація повідомлень

Синхронна реплікація повідомлень подібна до синхронної реплікації даних у запам'ятовувальних пристроях [3]. Один із вузлів, який обирається як первинний, забезпечує консистентність даних на всіх вузлах системи в такий спосіб:

- повідомлення про всі події, які відбуваються на будь-якому з вузлів системи, відправляються на первинний вузол;
- первинний вузол відправляє повідомлення про кожну подію на всі вузли системи;
- після отримання повідомлення від первинного вузла кожен із вузлів надсилає первинному вузлу підтвердження про отримання повідомлення;
- після того, як на первинний вузол надійдуть підтвердження від усіх вузлів системи про отримання його повідомлення, передача повідомлення вважається успішно завершеною.

Синхронну реплікацію повідомлень проілюстровано на рис. 3. Вузол A (первинний) відправляє повідомлення MSG_A на вузол B і чекає на повідомлення ACK_A, що підтверджує успішне отримання MSG_A вузлом B. Протягом часу очікування на ACK_A вузол A отримує повідомлення MSG_B від вузла B, але не реплікує його, доки передачу MSG_A не завершено. Після отримання ACK_A вузол A вважає передачу MSG_A завершеною і відправляє до вузла B раніше отримане повідомлення MSG_B. Після отримання ACK_B від вузла B передача MSG_B є завершеною.

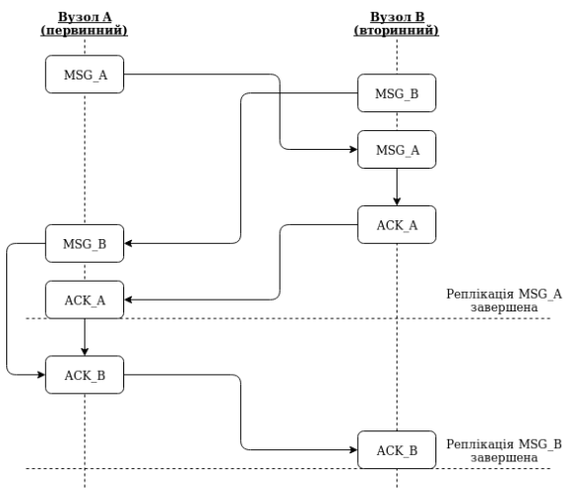

Рис. 3. Синхронна реплікація повідомлень

Порядок подій на обох вузлах виглядатиме ідентично:

```
Order = [ NONE, MSG_A, MSG_B ]
```

Вочевидь, при тому, що синхронна реплікація гарантує відтворення однакового порядку надходження повідомлень на всіх вузлах, вона є ефективною лише за умови наявності швидкісних каналів зв'язку між вузлами. У разі значної територіальної віддаленості вузлів один від одного, коли швидкість передачі даних між ними є обмеженою та є суттєві затримки передачі повідомлень, ефективність синхронної реплікації є низькою. Продуктивність вузлів знижується через значний час перебування в стані очікування. Синхронна реплікація також є дуже вразливою до втрати зв'язку між вузлами та виходу

з ладу якогось із вузлів. У подібних випадках потрібні спеціальні процедури для відновлення працездатності системи, інакше її функціювання буде заблоковано.

Асинхронна реплікація повідомлень

На відміну від синхронної реплікації, під час асинхронної реплікації повідомлень подібно до асинхронної реплікації даних [2] вузол, який відправляє повідомлення, не очікує підтвердження отримання, натомість продовжує роботу.

Асинхронну реплікацію проілюстровано на рис. 4. Вузол A відправляє повідомлення MSG_A до вузла B. Ще до того, як отримати повідомлення MSG_A, вузол B відправляє повідомлення MSG_B до вузла A. У результаті порядок повідомлень на вузлах A і B буде різним. Для забезпечення однаковості порядку повідомлення на всіх вузлах до повідомлень може бути додана часова відмітка – фактичний час створення повідомлення. У результаті повідомлення будуть однаково впорядковані на обох вузлах. Наявність годинника точного часу на вузлах є важливою вимогою для асинхронної реплікації, оскільки від точності часових відміток у повідомленнях залежить правильність упорядкування повідомлень.

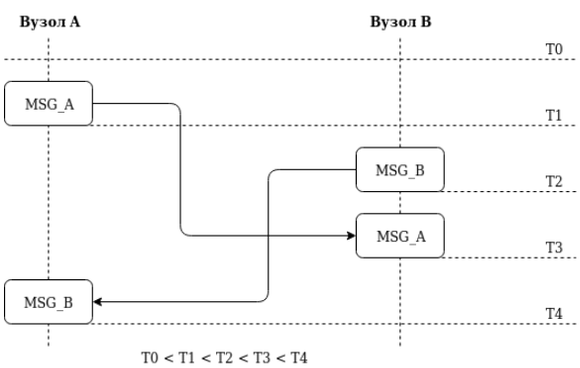

Рис. 4. Хронологічне впорядкування подій у системі з асинхронною реплікацією

Для формального запису порядку подій у системі з асинхронною реплікацією повідомлень скористаємося синтаксисом Python для словників (dictionaries):

```
Order = {
    T0:NONE,
    T1:MSG_A,
    T2:MSG_B
}
```

На відміну від синхронної реплікації, асинхронна реплікація здатна забезпечити ефективну взаємодію вузлів, рознесених на значну відстань, адже затримки передачі повідомлень не призводять до перебування вузлів у стані очікування.

Порівняно з синхронною реплікацією асинхронна реплікація також толерує тимчасову втрату зв'язку між вузлами. Повідомлення, адресовані вузлу, з яким втрачено зв'язок, утримуються в буфері і після відновлення зв'язку передаються до вузла адресата.

Зазначені фактори визначають те, що саме асинхронна реплікація переважно використовується в територіально розподілених мережах [1].

Утім, асинхронна реплікація повідомлень має певний недолік – вона не здатна забезпечити точний причинно-наслідковий порядок повідомлень на вузлах.

Причинно-наслідковий порядок подій

Хронологічний порядок вилаштовує послідовність подій згідно з часом, коли вони відбулися, проте не вказує на те, чи може певна подія бути реакцією на попередню подію. Згідно з рис. 4 подія, яка викликала створення повідомлення MSG_B, відбулася хронологічно пізніше, ніж подія, яка викликала створення повідомлення MSG_A. Але, оскільки в момент створення повідомлення MSG_B вузол B ще не отримав повідомлення MSG_A, MSG_B не може бути реакцією на MSG_A. Незважаючи на те, що повідомлення MSG_A було створено раніше, ніж повідомлення MSG_B, з погляду причинно-наслідкового порядку обидва повідомлення слід вважати такими, що були створені умовно одночасно – жодне з них не може вважатися реакцією на інше.

Сформулюємо ознаки причинно-наслідкового порядку подій у розподіленій системі:

- якщо в момент створення на вузлі B повідомлення MSG_B цей вузол уже отримав повідомлення MSG_A від вузла A, повідомлення MSG_B може вважатися реакцією на повідомлення MSG_A і в порядку подій має займати місце після MSG_A;
- якщо в момент створення на вузлі B повідомлення MSG_B цей вузол ще не отримав повідомлення MSG_A (яке хронологічно вже було створено на вузлі B), обидва повідомлення MSG_A та MSG_B у порядку подій мають поділяти одне місце як події, що відбулися умовно одночасно.

Розглянемо обмін повідомленнями між вузлами A і B (див. рис. 5). Вочевидь, повідомлення MSG1_B є реакцією на MSG1_A, а повідомлення MSG3_A є реакцією на MSG3_B. Повідомлення MSG2_A та MSG2_B не є реакцією одне на одне і вважаються такими, що створені умовно одночасно.

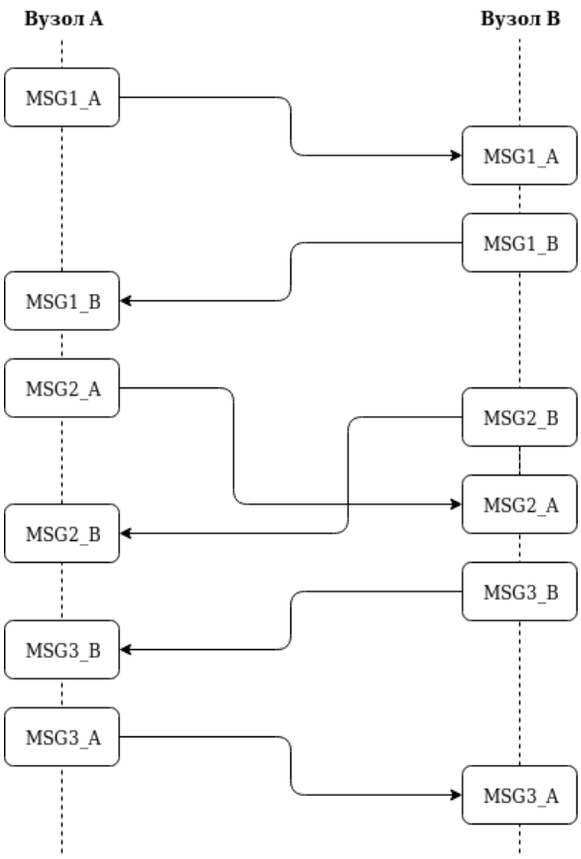

Рис. 5. Причинно-наслідковий порядок подій

Для кожного виду реплікації повідомлень з метою впорядкування повідомлень потрібно використати певний атрибут кожного повідомлення, значення якого визначає позицію в списку кожного повідомлення. Для синхронної реплікації таким атрибутом може бути просто порядковий номер. Для асинхронної реплікації атрибутом може бути часова відмітка – час створення повідомлення.

Розглянемо можливий варіант атрибута повідомлень у системах з причинно-наслідковим упорядкуванням. Потрібний атрибут повинен задовольняти такі вимоги:

- якщо повідомлення MSG_B є реакцією на повідомлення MSG_A, атрибут повинен визначити більш ранню позицію MSG_A порівняно з MSG_B;
- якщо жодне з повідомлень MSG_A та MSG_B не може бути реакцією на інше, тобто вони вважаються умовно одночасними, значення атрибута для обох повідомлень повинні бути однаковими.

Атрибутом, який задовольняє вказані вимоги, може бути порядковий номер позиції в списку подій подібно до синхронної реплікації. Але, на відміну від синхронної реплікації, на певних позиціях списку може перебувати не одне повідомлення, а декілька повідомлень, які вважаються умовно одночасно створеними на різних вузлах.

Приклад запису порядку подій у системі з причинно-наслідковим упорядкуванням повідомлень:

```
Order = {
    0:NONE,
    1:MSG1_A,
    2:MSG1_B,
    3:[ MSG2_A, MSG2_B ],
    4:MSG3_B,
    5:MSG3_A
}
```

Систему з причинно-наслідковим упорядкуванням подій можна вважати такою, що послідовно змінює свої дискретні стани. Умовою переходу системи з поточного стану в наступний стан є передача нових повідомлень. Це може бути одне повідомлення від певного вузла або декілька умовно одночасних повідомлень від різних вузлів. Перехід системи між станами під час передачі повідомлень згідно з рис. 5 ілюструє граф на рис. 6.

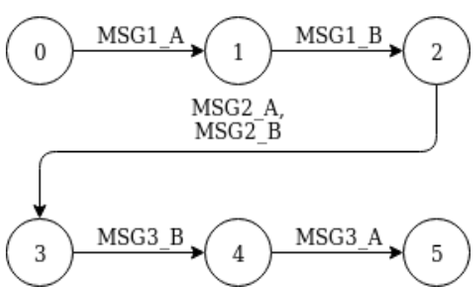

Рис. 6. Граф переходів системи між станами

Вочевидь, причинно-наслідкове впорядкування повідомлень може бути впроваджене в системі з будь-якою кількістю вузлів. Критично важливою вимогою до таких систем є здатність кожного вузла відправляти повідомлення всім вузлам системи. Відповідно, на кожному вузлі має бути інформація про всі інші вузли. Якщо система складається з великої кількості вузлів або якщо вузли в системі додаються і вилучаються динамічно, виконання зазначеної вимоги може бути ускладнене. На рис. 7 проілюстровано передачу повідомлень у системі з 3 вузлів, граф переходів системи між станами наведено на рис. 8.

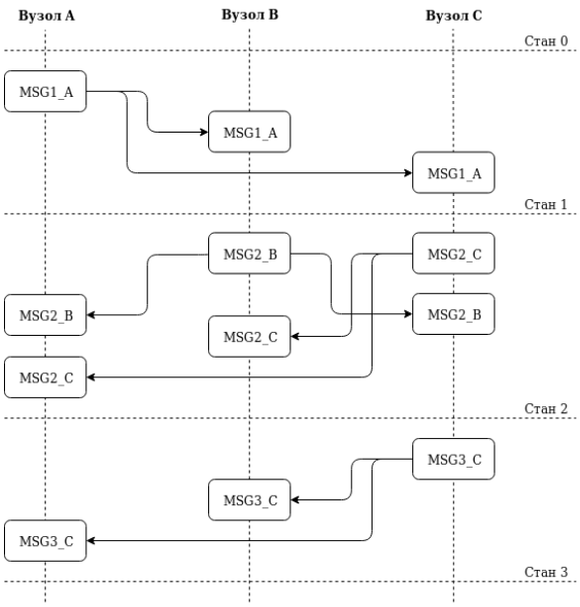

Рис. 7. Передача повідомлень у системі з 3 вузлів

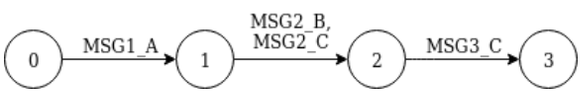

Рис. 8. Граф переходів між станами системи з 3 вузлів

Відповідний запис порядку подій виглядає так:

```
Order = {
    0:NONE,
    1:MSG1_A,
    2:[ MSG2_B, MSG2_C ],
    3:MSG3_C
}
```

Висновки

Для повної синхронізації вузлів територіально розподіленої системи з вузлами, що рознесені на значну відстань один від одного, важливою задачею є відтворення причинно-наслідкового порядку подій. Синхронна реплікація повідомлень розв'язує цю задачу, але придатна лише в зосереджених системах. Асинхронна реплікація здатна відновити хронологічний порядок подій, але не забезпечує збереження їх логіки – факт подій, які стали реакцією на інші події. Причинно-наслідкове впорядкування використовує поняття умовно одночасних подій і базується на визначенні дискретних станів системи, між якими вона переходить унаслідок подій, що відбуваються.

Список використаної літератури

1. Arregoces M. Data Center Fundamentals / Mauricio Arregoces, Maurizio Portolani. – Sixth Printing. – Cisco Press, 2009.
2. Goel S. Data Replication Strategies in Wide-Area Distributed Systems [Electronic resource] / Sushant Goel, Rajkumar Buyya. – Mode of access: http://www.cloudbus.org/papers/DataReplicationInDSChapter2006.pdf. – Title from the screen.
3. Sivasankar P. Synchronous (vs) Asynchronous Replication [Electronic resource] / P. Sivasankar. – Mode of access: https://www.vembu.com/blog/synchronous-vs-asynchronous-replication. – Title from the screen.
4. Synchronous vs. asynchronous replication [Electronic resource]. – May 1, 2016. – Mode of access: http://cloudbasic.net/white-papers/synchronous-vs-asynchronous-replication. – Title from the screen.

Dmytro Cherkasov

REASON-CONSEQUENCE EVENTS ORDERING ON THE NODES OF WIDE AREA DISTRIBUTED SYSTEMS

Using the Internet as a universal and global media for information exchange apart from convenience has brought up a set of significant challenges. To secure scalability and redundancy and to provide service over wide geographic areas, information systems include multiple distributed nodes that are placed in the best way to optimize data flows and bring the service as close to consumers as possible. One of critically important tasks when designing such systems is synchronization between nodes of the system, which means bringing them to a coordinated state and replicating data between them.

The task of synchronization may be considered as message exchange between the nodes, as a result of which each of the nodes gets into the desired state. Due to message propagation delays, which are common in wide area systems, it is possible that reason-consequence relations between events are lost. While a correct chronological order of the events can be achieved by using timestamps for the messages, the fact is that that some message appeared as a reaction to some other message that may be gone. To preserve not only the chronological order but also the logic of the events during synchronization of nodes in a wide area system, it is necessary to use reason-consequence events ordering.

The other problem with the distributed system is handling the case of connectivity loss between parts of the system known as split brain. Choosing one of disconnected parts as a single operational one and considering all the others as failed simplifies synchronization upon recovery but may cause a severe performance degradation. On the other hand, allowing all of the disconnected parts to operate significantly complicates integration of operational results of all the nodes. Adding reason-consequence relations between events may lead to more natural ordering of events that occurred on disconnected nodes, and simplify synchronization.

Keywords: synchronization, distributed system, wide area system, synchronous replication, asynchronous replication, reason-consequence ordering.

Матеріал надійшов 15.05.2018