ІНФОКОМУНІКАЦІЙНІ ТЕХНОЛОГІЇ ТА ЕЛЕКТРОННА ІНЖЕНЕРІЯ
INFORMATION AND COMMUNICATION TECHNOLOGIES, ELECTRONIC ENGINEERING

№ 2 (4), 2022

https://doi.org/10.23939/ictee2022.02.058

МЕТОД ДОСЯГНЕННЯ КОНСЕНСУСУ ВРОЗПОДІЛЕНИХ СЕРВІСНИХ СИСТЕМАХ

С.Журавель, О. Шпур, Ю. Пиріг

Національний університет "Львівська політехніка", вул. С. Бандери, 12, Львів, 79013, Україна

Відповідальний за рукопис: Станіслав Журавель (e-mail:stanislav.s.zhuravel@lpnu.ua)

(Подано 10 Листопада 2022)

У статті розглядаються проблеми досягнення консенсусу врозподілених інфокомунікаційних системах для підвищення відмовостійкості їхньої роботи. Запропоновано метод досягнення консенсусу системи шляхом зменшення проблем лінеаризованості та повного порядку повідомлень і мережі.Запропоновано виявлення збоїв вузлів у кластері. Суть методу полягає у вирішенні проблеми консенсусу через алгоритм трансляції повного порядку шляхом використання операції лінійного приросту, що дозволить стверджувати про правильну роботу (консенсус) розподіленої системи. Подальший аналіз системи під час коректної роботи виявляє несправності за часом тайм-аут, який сигналізує про втрату зв'язку з вузлом, з яким неможливо обмінюватися повідомленнями. Такий підхід дозволить виявляти несправність вузла та уникнути можливих помилкових висновків про його несправність. Для визначення очікуваної затримки пропонуємо використати модель SARIMA. Її використання дозволить проводити аналіз даних із використанням функцій авторегресії та ковзного середнього. Це, своєю чергою, зменшить кількість реконфігурацій системи та виборів нового керівника і, як наслідок, підвищить відмовостійкість розподіленої системи.

Ключові слова: *розподілені інформаційні системи, алгоритм консенсусу, аналіз даних.*
УДК 621.126

1. Вступ

Сьогодні більшість інфокомунікаційних систем є розподіленими. Такі системи, як правило, функціонують в умовах можливих збоїв мережі, а відтак –потребують можливості зберігати працездатність навіть в умовах відмов частин системи. Однак наявність великої кількості вузлів, як окремо взятих елементів системи, породжує проблему розсинхронізації її роботи [1]. Всі ці можливості так чи інакше можуть бути зведені до вирішення проблеми консенсусу в розподіленій системі. Для вирішення проблеми консенсусу було розроблено велику кількість алгоритмів консенсусу, водночас, майже всі реалізації страждають від одних і тих же проблем.

2. Формалізація проблематики досліджень

Проблема консенсусу зазвичай формалізується наступним чином: один або кілька вузлів можуть запропонувати значення, і алгоритм консенсусу приймає рішення щодо одного з цих значень. Прикладом може слугувати бронювання місць, коли кілька клієнтів одночасно намагаються купити останнє місце на літаку, кожен вузол, який обробляє запит клієнта, може запропонувати ідентифікатор клієнта, якого він обслуговує, і рішення вказує, хто з цих клієнтів отримав місце.

У цьому формулюванні алгоритм консенсусу повинен задовольняти такі властивості [2]:

- Узгодженість – немає двох вузлів, які б вирішили по-різному.
- Цілісність – жоден вузол не приймає рішення двічі.
- Вірність – якщо вузол визначає значення v, то v є запропонованим значенням (існує в основному для того, щоб виключити тривіальні рішення. Наприклад, можна створити алгоритм, який завжди вирішує нуль, незалежно від того, що було запропоновано).
- Завершеність – кожен вузол, який є в працездатному стані, в кінцевому підсумку пропонує певне значення.

Найбільш відомі алгоритми консенсусу включають Viewstamped Replication (VSR), Paxos [2], Raft [3], та Zab [4] (розглянемо детальніше їх пізніше). Більшість із цих алгоритмів фактично не використовують описану тут формальну модель (пропонування та прийняття рішення щодо єдиного значення, задовольняючи властивості моделі). Замість цього вони вибирають послідовність значень, що робить їх алгоритмами широкомовної передачі повного порядку.

Загалом досягнення консенсусу в розподіленій системі означає, що декілька вузлів мають дійти згоди у якомусь питанні.

3. Алгоритми досягнення консенсусу та їх невирішені проблеми

Як вже згадувалось, найбільш відомі алгоритми консенсусу включають Paxos [2] та його наступника Raft [3].

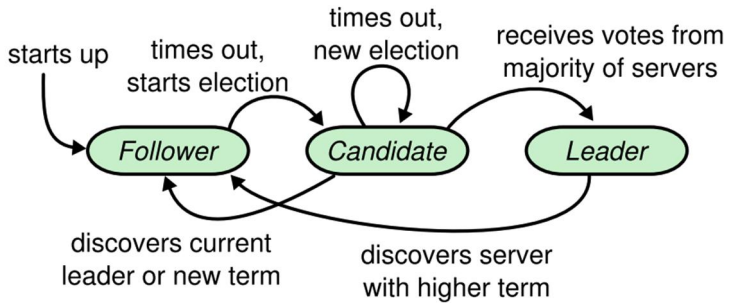

Рис. 1. Процес вибору лідера в кластерів відповідно до роботи алгоритму Raft [3]

Raft обирає лідера в кластері, який диктує всім іншим які команди/повідомлення виконати/зберегти а також надсилає повідомлення "ритму" всім учасникам кластеру для підтвердження своєї присутності та працездатності (рис. 1). Сам процес вибору лідера відбувається шляхом голосування, яке пропонується одним з учасників кластеру. Як тільки він перестане отримувати повідомлення ритму від лідера – лідер переобирається. Новим лідером стає кандидат, який отримав голоси більшості кластера.

Paxos, своєю чергою, працює без обрання лідера (так званий "безлідерний" алгоритм). Під час його функціонування не вимагається наявність лідера, проте він є значно складнішим в реалізації. Функціонування алгоритму Paxos виглядає наступним чином: вузол Paxos може виконувати будь-яку або всі з трьох ролей: того хто пропонує значення, того хто підтверджує запропоноване значення (приймач) та учня (рис. 2).

Один з вузлів пропонує значення, яке він хоче узгодити з кластером (щоб досягнути консенсусу системи). Для цього вузол надсилає пропозицію, всім вузлам які виступають в ролі "приймачів", які вирішують, чи приймати це значення чи ні. Кожен приймач самостійно обирає значення, а також отримати декілька пропозицій, кожну від різних серверів. В результаті послідовного перебору, надсилає своє рішення учням, які визначають, чи було досягнуто консенсусу в виборі значення чи ні. Щоб значення було прийняте у Paxos, більшість приймачів повинні вибрати те саме значення.

Paxos, як і Raft, є складними алгоритмами з великою кількістю специфічних ситуацій та технік з їх вирішення. Розглянемо найпростішу ситуацію, коли всі приймачі погоджуються із вибором значення (рис. 3).

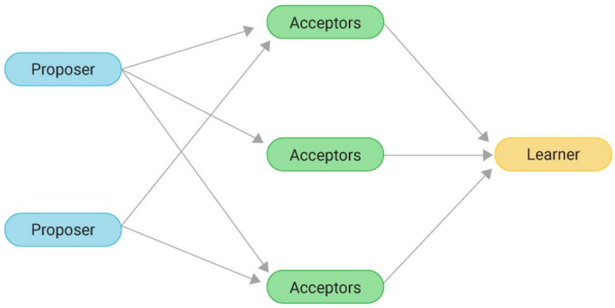

Рис. 2. Принцип роботи алгоритму Paxos [2]

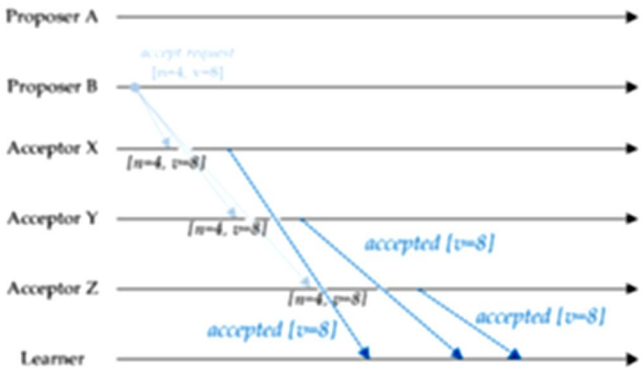

Рис. 3. Раунд алгоритму Paxos, в якому всі вузли погоджуються з запропонованим значенням

Процес, за допомогою якого вузли голосують за прийняття рішення, є свого роду синхронною реплікацією. Проведемо паралель з базами даних які часто налаштовані на використання асинхронної реплікації. У цій конфігурації деякі дані, які вважались надійно збереженими, потенційно можуть бути втрачені під час відмови одного з вузлів системи, але багато інженерів вирішують прийняти цей ризик заради кращої продуктивності системи.

Для досягнення консенсусу системи і, як наслідок, підвищення продуктивності її функціонування, завжди потрібна більшість. Це означає, що потрібно мінімум три вузли, щоб витримати один збій (решта – два з трьох, формують більшість), або мінімум п’ять вузлів, щоб витримати дві відмови (решта – три з п’яти, утворюють більшість). Якщо збій мережі відрізає деякі вузли від більшості, лише більша частина вузлів разом може досягти прогресу, а решта, відрізана збоями в мережі, блокується.

Більшість алгоритмів консенсусу припускають фіксований набір вузлів, які беруть участь у голосуванні, що означає, що не можна просто додати чи видалити вузли в кластері. Існують розширення (Vertical Paxos [7], Raft’s joint consensus [3], Dynamic Plain Paxos [8]) до алгоритмів консенсусу, які дозволяють змінювати кількість вузлів у кластері з часом, але вони менш зрозумілі та продуктивні, ніж алгоритми статичного членства.

Для досягнення консенсусу в системі зазвичай покладаються на затримки для виявлення несправних вузлів. У середовищах із дуже мінливими мережевими затримками, особливо в географічно розподілених системах, часто буває, що вузол помилково вважає, що лідер вийшов з ладу через тимчасову проблему мережі. Хоча ця помилка не завдає шкоди роботі системи в загальному, але це

призводить до переобрання лідера, оскільки інша частина системи помилково вважає, що лідер не є працездатним. З іншого боку, часте переобрання лідера призводить до просідання продуктивності системи загалом, оскільки система може витрачати більше часу на вибір лідера, ніж на виконання будь-якої корисної роботи.

Іноді алгоритми консенсусу особливо чутливі до проблем мережі. Наприклад, в результаті роботи алгоритму Raft [9–11] може настати випадок, коли вся мережа працює правильно, за винятком одного конкретного мережевого зв'язку, який постійно підводить систему. При цьому відбувається постійна зміна лідера, що провокує додаткові затримки в роботі всієї системи, а відтак, система фактично ніколи не прогресує.

4. Метод досягнення консенсусу алгоритмів у розподілених сервісних системах

Проаналізувавши проблеми алгоритмів досягнення консенсусу, можна спостерігати зменшення ефективності їхньої роботи внаслідок недосконалості виявлення несправності мережі або окремих вузлів системи [1]. Найчастіше несправності мережі виявляють через перевищення вузлом тайм-ауту (T_{delay}), що сигналізує про втрату комунікації з вузлом, з яким не можливо обмінятися повідомленнями. Для своєчасного виявлення несправності мережі, яка може виникнути на етапі функціонування, у дослідженні [1]запропоновано алгоритм. Основні його кроки представлено на рисунку 4.

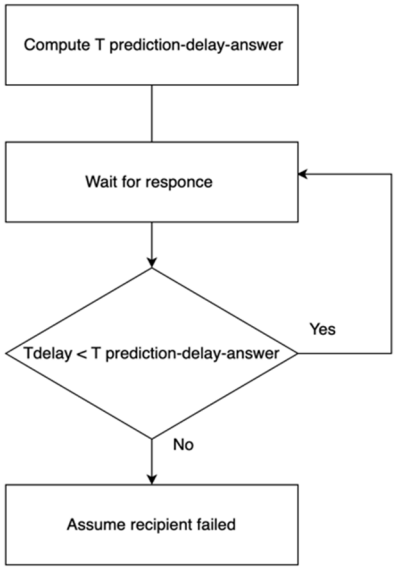

Рис. 4. Блок-схема алгоритму виявлення несправностей [1]

Пропонуємо проводити прогнозування затримки-відповіді вузлів $T_{prediction\text{-}delay\text{-}answer}$. В основі такого прогнозування лежить статистичний аналіз даних роботи системи впродовж часу її функціонування. Під часом функціонування будемо розуміти досягнення консенсусу системи в наслідок роботи алгоритму, включаючи час, коли алгоритм проводить реконфігурацію системи. Маючи множину значень $T_{prediction\text{-}delay\text{-}answer}$ кожен вузол приймає рішення про несправність вузла коригуючи T_{delay}. Коригування T_{delay} повинно забезпечувати виконання умови представленої у формулі 1:

$$T_{delay} < T_{prediction-delay-answer}. \quad (1)$$

Такий підхід дасть змогу покращити висновок системи про несправність того чи іншого вузла та ухилитись від можливих хибних висновків про його несправність. Це, чергу своєю чергою, до-

зволить зменшити час на реконфігурацію системи та обрання нового лідера та підвищити відмовостійкість розподіленої системи.

Для визначення очікуваної затримки пропонуємо використати модель SARIMA (Seasonal Autoregressive Integrated Moving Average). Розглянемо компоненти моделі.

Компонент авторегресії моделі SARIMA представлений $AR(p)$, де параметр p визначає кількість серій із запізнення, які ми використовуємо, і який представлений формулою.

$$AR(p) = c + \sum_{n=1}^{p} a_n y_{n-t} + e_t \ . \tag{2}$$

Водночас, більше значення параметра p означає більше повернення назад у часі. Процес авторегресії $AR(1)$ – це процес, у якому поточне значення базується на безпосередньо попередньому значенні, тоді як процес $AR(2)$ – це процес, у якому поточне значення базується на двох попередніх значеннях. Процес $AR(0)$ використовується для білого шуму і не має залежності між термінами.

Також введемо в модель процес ковзного середнього q-порядку, який позначений $MA(q)$ та визначається формулою:

$$MA(q) = c + e_i + \theta_1 e_{i-1} + ... + \theta_q e_{i-q} \ . \tag{3}$$

Замість того, щоб використовувати минулі значення змінної прогнозу в регресії, модель ковзного середнього використовує попередні помилки прогнозу в моделі. В аналізі часових рядів модель ковзного середнього (MA-модель), також відома, як процес ковзного середнього, є поширеним підходом для моделювання однофакторних часових рядів. Модель ковзного середнього визначає, що вихідна змінна перехресно корельована з неідентичною собі випадковою змінною.

У статистичному аналізі часових рядів моделі авторегресії та ковзного середнього (разом ARMA) забезпечують економний опис (слабко) стаціонарного стохастичного процесу в термінах двох поліномів, один для авторегресії (AR), а другий – для ковзного середнього (MA).

Водночас, I (Integrated) представляє різницю необроблених спостережень, щоб часовий ряд став стаціонарним (тобто, значення даних замінюються різницею між наявними значеннями даних і попередніми). Значення стаціонарності відіграє значну роль, оскільки, стаціонарний часовий ряд – це ряд, властивості якого не залежать від часу спостереження за ним. Отже, часові ряди з тенденціями або сезонністю не є стаціонарними – тренд і сезонність впливатимуть на значення часового ряду в різні часи. Така обробка дасть можливість точніше передбачати значення.

Модель SARIMA розширює модель ARIMA, враховуючи сезонність. Такі моделі виражаються як $(p,d,q)\times(P,D,Q)_m$, де (p,d,q) є для моделі ARIMA, тоді як $(P,D,Q)_m$ виражають сезонну авторегресію, інтеграцію та компоненти ковзного середнього, де період сезонності дорівнює m.

4.1. Аналіз даних із використанням функцій авторегресії та ковзного середнього

Для доведення адекватності роботи з обраною моделлю та аналізу даних із використанням функцій авторегресії та ковзного середнього обрано програмний пакет Pandas. Pandas – це швидкий, потужний, гнучкий і простий у використанні інструмент аналізу та обробки даних з відкритим кодом.

Для аналізу було обрано сет даних, який містить час відповіді віддаленого сервера впродовж тижня, де час між двома послідовними запитами становив 10 секунд. Обраний датасет представлений на рис. 5.

Як бачимо, дані містять значні стрибки в часі відповіді, що, очевидно, говорить про несправність обладнання в такі моменти, оскільки нас цікавить час затримки мережі. Відфільтруємо значення, які значно перевищують норму, водночас для спрощення моделювання обробимо дані так, щоб вони включати медіану часу відповіді впродовж години, після оброблення отримаємо наступний сет даних (рис. 6).

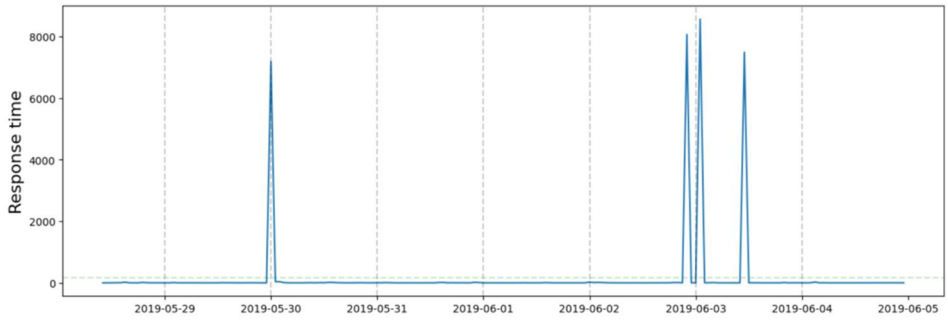

Рис. 5. Необроблений вхідний сет даних, (час очікування відповіді сервера напротязі 05.29–06.05)

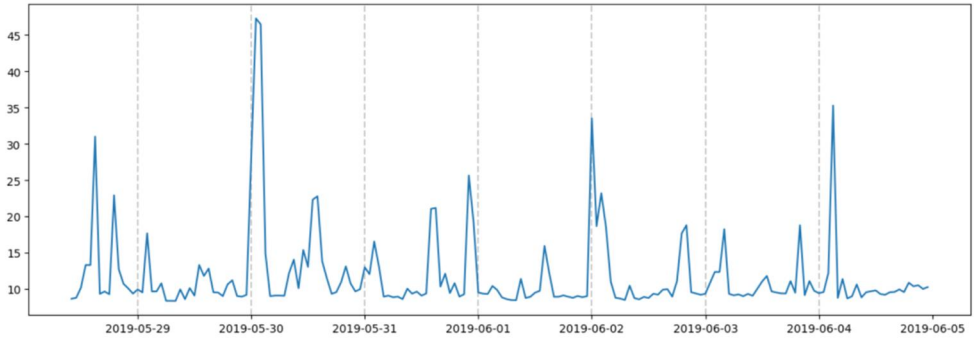

Рис. 6. Відфільтрований та оброблений вхідний сет даних (час очікування відповіді сервера напротязі 05.29–06.05)

Для визначення авторегресійної частини нашої моделі (AR) скористаємося графіком автокориляційної функції (ACF).

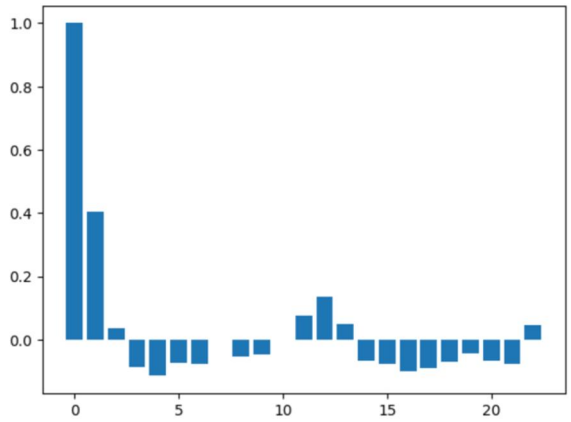

Рис. 7. Графік автокореляційної функції (ACF)

Автокореляція – це кореляція між часовим рядом та історичною версією цього ж ряду, тобто, кореляція між спостереженнями в поточній точці часу та спостереженнями в попередніх точках часу. Функція автокореляції в точці 0 є кореляцією часового ряду з самим собою, і тому призводить до кореляції 1.

Як видно з графіка, існує сильна залежність в точці 1, далі ж залежність суттєво спадає. На основі цього можна сказати, що MA-компонент нашої моделі може мати значення 1, тим не менше, для точнішого моделювання скористаємося MA(14), оскільки в цих точках значення ACF залишається доволі великим.

Для визначення степеня AR-компоненти побудуємо графік PACF (Partial auto correlation function).

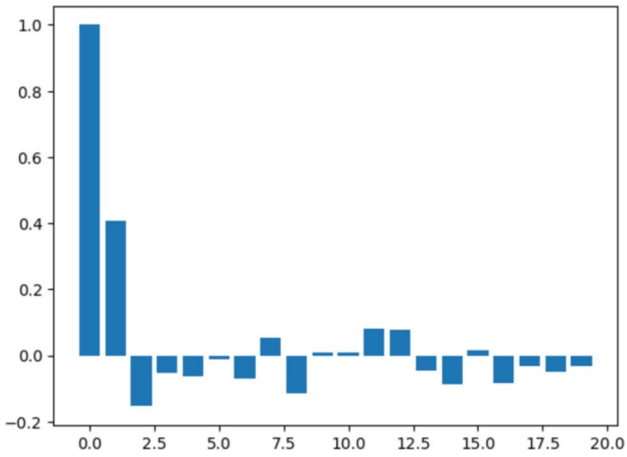

Рис. 8. Графік часткової автокореляційної функції (PACF)

Функція часткової автокореляції (PACF) подібна до ACF, за винятком того, що вона відображає лише кореляцію між двома спостереженнями, яку затримки між цими спостереженнями не пояснюють. Наприклад, часткова автокореляція для відставання 3 є лише кореляцією, яку не пояснюють відставання 1 і 2. Іншими словами, часткова кореляція для кожного відставання є унікальною кореляцією між цими двома спостереженнями після часткового видалення проміжних кореляцій.

Проаналізувавши графік PACF, можемо зробити висновок, що значення в точках 1, 2, 3, 4 залишаються на доволі високому рівні, а отже, для побудови авторегресивної частини (AR) нашої моделі буде достатньо цих значень, тобто AR(4), тим не менше, для побудови більш точного передбачення, як і в випадку ACF, скористаємося AR(14), оскільки значення в цих точках залишаються на досить високому рівні.

Також введемо інтеграцію (I) даних в нашу модель, щоб отримати стаціонарний ряд значень з метою отримання кращого прогнозу, як згадувалось вище.

Розглянувши вхідний сет значень можна побачити сезонну частину даних, тобто відслідковується, хоч і недостатньо чітко, залежність зміни функції в продовж доби. Це означає що можна покращити нашу модель ввівши сезонність, тобто скористатись SARIMA – Seasonal Auto Regressive Integrated Moving Average model. Підібрати сезонність для нашої моделі досить просто, оскільки сезонність прослідковується впродовж доби, можна сказати що сезон становитиме 24, тобто 24 години. Сезонність також можна прослідкувати за допомогою графіка ACF.

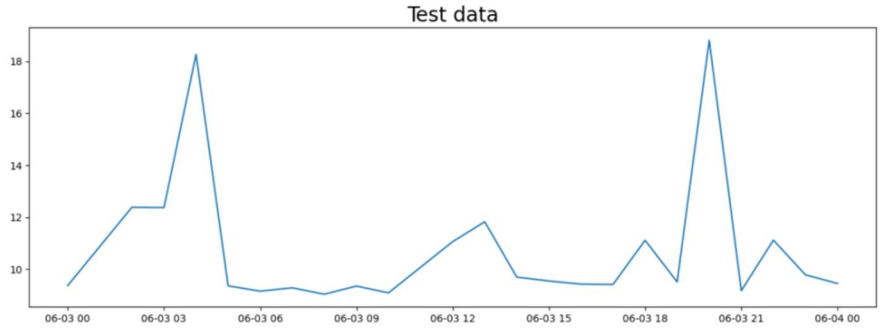

Рис. 9. Тестовий набір даних, розподіл часу відповіді сервера впродовж 06.03–06.04

Як результат розгляду даних та функцій ACF та PACF визначено модель S(1,1,1,24) AR(14)I(1)MA(14), де значення 1,1,1 – значення AR, I та MA в сезонній складовій відповідно.

Для тренування та тестування нашої моделі розділимо вхідний сет даних на дві частини, частину для тренування та тестування відповідно. Дані від 29.05 до 3.06 відокремлено для тренування моделі, відповідно, дані від 3.06 до 4.06 потраплять в збірку для тестування моделі, тобто для порівняння з передбаченими значенням. Тестовий набір даних представлено на рис. 9.

5. Дослідження ефективності методу досягнення консенсусу алгоритмів з використанням функцій авторегресії та ковзного середнього

Застосувавши представлену модель аналізу даних із використанням функцій авторегресії та ковзного середнього та описаний у [1] алгоритм виявлення несправностей, отримано ймовірнісний розподіл затримки відповіді сервера впродовж дня (рис. 10).

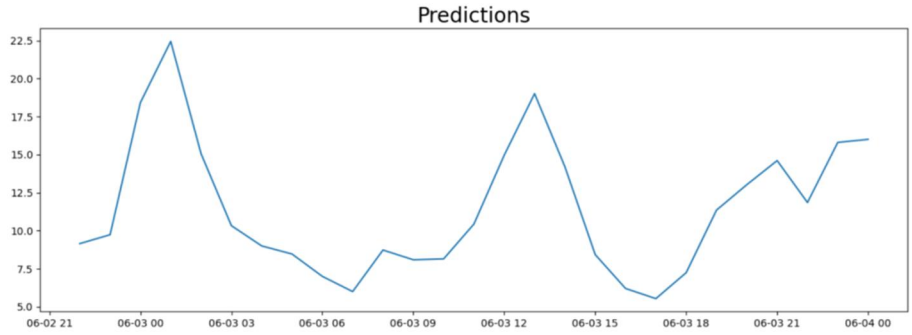

Рис. 10. Передбачений набір даних, розподіл часу відповіді сервера впродовж 06.03–06.04

Розглянемо детальніше тестовий та передбачений набори даних (рис. 11).

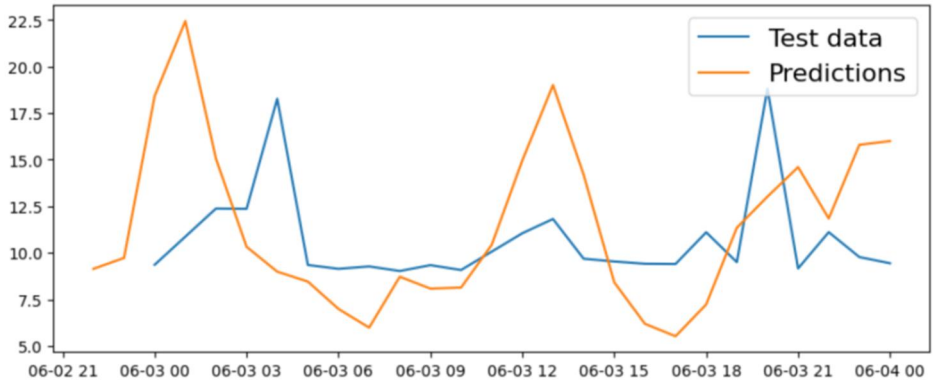

Рис. 11. Передбачений та тестовий набори даних, розподіл часу відповіді сервера впродовж 06.0 –06.04

Як бачимо, передбачені значення повторюють форму тестових даних, а різниця між дійсним і передбаченим значенням не перевищує 8 мс. Отже, можна стверджувати що наш метод є ефективним, однак необхідно продовжити пошук інших моделей для аналізу затримки та провести порівняльний аналіз роботи різних моделей передбачення. Водночас, можна спробувати скоригувати нашу модель, ввівши інші коефіцієнти відповідних частин.

Повертаючись до формули 1, маючи $T_{prediction\text{-}delay\text{-}answer}$, отриманий за допомогою нашої моделі, можна скоригувати роботу алгоритму консенсусу, що в результаті дозволить зменшити час на реконфігурацію системи та обрання нового лідера та підвищить відмовостійкість розподіленої системи в цілому.

Висновки

У цій статті розглядаються проблеми розподілених алгоритмів та пропонуєтсья їх вирішення за допомогою часового аналізу. Є багато речей, які можуть піти не так у розподілених системах, що може призвести до збою системи. Рішення полягає побудові системи, яка могла б витримати проблеми, що виникають під час її функціонування. Виявляється, наявність алгоритму, здатного досягти консенсусу, є надзвичайно важливою для систем, які хочуть функціонувати належним чином, незважаючи на збої в мережі. Хоча консенсус опущено в системах, які віддають перевагу продуктивності, вони все ще сильно залежать від систем, які реалізують для них алгоритми консенсусу (наприклад, Zookeeper тощо), щоб впоратися із завданням, яке зводиться до консенсусу, в той же час, маючи деяку слабшу модель узгодженості. Водночас, наявні сьогодні алгоритми мають низку проблем, вирішення яких значно збільшить продуктивність алгоритмів і, як результат, систем, які їх використовують. У цій статті розглянуто проблеми, які постають в наявних імплементаціях, та представлено підхід і модель аналізу даних для вирішення однієї з проблем алгоритмів.

Надалі продовжимо пошук моделей, за допомогою яких можна було б передбачити значення затримки відповіді сервера, а також спробуємо застосувати отриману модель до більшого сету даних з метою підтвердження та покращення результатів. Також є необхідність провести оцінку стабільності роботи алгоритму консенсусу з використанням запропонованого рішення.

Список використаних джерел

[1] *Zhuravel S., Klymash M., Shpur O. and Lavriv O. "Achieving Consistency and Consensus of Distributed Infocommunication Systems", 16th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET). Pp. 386–389, February 22–26, 2022.*

[2] *Lamport L. "The Part-Time Parliament", ACM Transactions on Computer Systems. Vol. 16. Pp. 133–169, May 1998.*

[3] *Ongaro D. and Ousterhout J. "Search of an Understandable Consensus Algorithm", at USENIX Annual Technical Conference (ATC), 2014.*

[4] *Junqueira F. P. and Reed B. in ZooKeeper: Distributed Process Coordination. O'Reilly Media, 2013.*

[5] *Chandra T. D., and Toueg S. "Unreliable Failure Detectors for Reliable Distributed Systems", Journal of the ACM. Vol. 43. Pp. 225–267, March 1996.*

[6] *Lamport L. "Paxos Made Simple", 2001.*

[7] *Lamport L. "Vertical Paxos and Primary-Backup Replication", 2009.*

[8] *Rystsov D. "Dynamic Plain Paxos", 2015.*

[9] *Howard H. and Crowcroft J. "Coracle: Evaluating Consensus at the Internet Edge", in Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), 2015.*

[10] *Balakrishnan M., Malkhi D. and Wobber T. "Tango: Distributed Data Structures over a Shared Log", in 24th ACM Symposium on Operating Systems Principles (SOSP), 2013.*

[11] *Kleppmann M. Designing Data-Intensive Applications, O'Reilly UK Ltd., 2017.*

METHOD OF ACHIEVING CONSENSUS IN DISTRIBUTED SERVICE SYSTEMS

S. Zhuravel, O. Shpur, Yu. Pyrih

Lviv Polytechnik National University, 12, S. Bandery Str., Lviv, 79013, Ukraine

This article examines the problems of distributed algorithms and proposes their solution using temporal analysis. There are many things that can go wrong in distributed systems that can cause the system to crash. The solution to this is to build a system that can withstand the problems that arise during its operation. It turns out that having an algorithm capable of reaching consensus is extremely important for systems that want to function properly despite network failures. Although consensus is omitted in performance-oriented systems, they still rely heavily on systems that implement consensus algorithms for them (such as Zookeeper, etc.) to handle the consensus-reduced task, while at the same time having some weaker consistency model. In turn, the algorithms available today have several problems, the solution of which will significantly increase the performance of the algorithms and, as a result, the systems that use them. This article discusses the problems that arise in existing implementations and presents a data analysis technique and model for solving one of the algorithm problem.

Keywords: *Distributed information systems, consensus algorithm, data analysis.*