Сучасний стан наукових досліджень та технологій в промисловості. 2024. №2 (28)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

УДК 004.94
DOI: https://doi.org/10.30837/2522-9818.2024.2.065

Д. Гольдінер

ЗАСТОСУВАННЯ МОВИ ПРОГРАМУВАННЯ *GO* ДЛЯ МОДЕЛЮВАННЯ ПРОЦЕСІВ МАСОВОГО ОБСЛУГОВУВАННЯ

Предметом дослідження статті є методи та підходи до програмного моделювання систем масового обслуговування на прикладі багатоканальної системи з обмеженою чергою та відмовами в разі її переповнення. **Мета роботи** – обґрунтування доцільності застосування сучасних комп'ютерних інформаційних технологій, а саме мови програмування *Go* для моделювання систем масового обслуговування. У статті вирішуються такі **завдання**: формулювання досліджуваної системи масового обслуговування; визначення компонентів, критеріїв спорідненості систем масового обслуговування з їх програмними моделями; загальний огляд конкаренсі як математичної моделі; опис підходів та інструментарію мови програмування *Go*. Упроваджуються такі **методи**: мова програмування *Go* та її інструментарій, конкаренсі, паралельне виконання. **Досягнуті результати**: сформульовано досліджуване завдання масового обслуговування; визначено критерії порівняння компонентів систем масового обслуговування з інструментарієм мови програмування *Go*; проаналізовано доцільність використання мови програмування *Go* для моделювання систем масового обслуговування; подальшого розвитку набув інструментарій для комп'ютерного моделювання систем масового обслуговування; запропоновано застосування підходів конкаренсі, їх імплементації в мові програмування *Go* до моделювання систем масового обслуговування. **Висновки**. Мова програмування *Go* є дуже вдалою технологією для моделювання систем масового обслуговування. Її філософія, спосіб роботи, а також вбудований інструментарій має широкі можливості для моделювання різноманітних систем масового обслуговування. Використання зазначеної мови є доцільним і дозволяє наблизити поведінку програми до модельованого процесу, спростити імплементацію та зменшити час, необхідний на оброблення даних. Визначено значні перспективи щодо подальшого впровадження програмного продукту, реалізованого мовою *Go* у сфері моделювання процесів масового обслуговування.

Ключові слова: комп'ютерне моделювання; система масового обслуговування; мова програмування *Go*; конкаренсі.

Вступ

Останнім часом кількість навантажених систем, що обслуговують потоки заявок, які описуються випадковими законами, неперервно зростає. Така тенденція спостерігалася завжди, але тривалий час постійне вдосконалення одноядерних процесорів завдяки збільшенню кількості транзисторів перекривало потреби. Однак після певного періоду збільшувати продуктивність окремих ядер процесорів стало вкрай проблематично – закон Мура перестав діяти. Логічним рішенням стала ідея паралельного виконання завдань на окремих ядрах процесора та збільшення загальної продуктивності пристрою таким чином. Унаслідок цього виробники почали звертати увагу конструкторів на збільшенні енергоефективності ядер, а також на одночасному розміщенні більшої їх кількості на одному чипі. Одноядерні процесори були майже повністю витіснені з ринку багатоядерними аналогами.

Утім, сам по собі багатоядерний процесор не має жодних переваг, оскільки для ефективного використання його потенціалу необхідна підтримка багатопотоковості тим програмним забезпеченням, що будуть на ньому виконуватись, разом з операційною системою. І під час спроби запустити на такому процесорі програму, що використовує лише один потік, спостерігатимемо такі самі показники швидкодії, як і в ситуації з одноядерним процесором.

З появою багатоядерних процесорів обсяги запитів почали зростати із значно більшою швидкістю, ніж можливості з масштабування ресурсів, що використовуються в обробленні даних. Водночас поновлюється запит на оптимізацію програмного продукту для більш ефективного застосування наявних ресурсів.

Одним із розділів математики, що описує поведінку процесів з оброблення значної кількості запитів із подальшим паралельним опрацюванням, є теорія масового обслуговування. Її активний розвиток припав на 50–70 рр. XX ст., що було пов'язано з індустріалізацією та розвитком телефонії. Завдання, зумовлені потребами виробництва та комунікації, спонукали до створення математичного апарату, що дасть змогу оптимізувати процеси. Можна стверджувати, що з поширенням технології

© Д. Гольдінер, 2024

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 2 (28)

Інтернет експоненційним зростанням кількості мережних запитів та обсягів даних, які постійно потребують оброблення, теорія масового обслуговування може набути нового розвитку. Вона буде застосована до подолання чергових викликів, сформованих потребами XXI ст.

Аналіз наявних підходів до розв'язання завдання

Система масового обслуговування (СМО) – є складником теорії ймовірностей та математичної статистики, що використовуються для аналізу процесів, в яких у випадкові моменти часу надходять вимоги для подальшого їх опрацювання наявними каналами обслуговування. Математичне моделювання систем, які мають велику інтенсивність надходження вимог, що є непропорційною продуктивності їх оброблення, дає змогу оцінити потенціальну пропускну здатність, а також її збільшити. Застосування стохастичного аналізу дозволяє аналізувати час очікування заявок і визначати оптимальну кількість каналів оброблення для задоволення потреб [1].

Основні елементи СМО:

– вхідний потік вимог – це послідовна сукупність вимог (заявок) на надання певної послуги, що надходить до системи. Вхідний потік може мати різні види ймовірнісного розподілу залежно від потреб та особливостей процесу, що моделюється. Зазвичай параметризується через інтенсивність прибуття λ заявок, що впливає на завантаженість системи, час очікування та ймовірність відмови в обслуговуванні;

– черга – це механізм для тимчасового зберігання заявок, які очікують на обслуговування. Черга може мати обмежену або необмежену ємність. Структура та політика черги щодо новоприбулих заявок відіграють важливу роль у визначенні порядку опрацювання заявок. Це значно впливає на ймовірність відмови в разі надходження в переповнену чергу, а також на розподілення навантаження поміж каналами;

– канали обслуговування – технічні пристрої або люди, які забезпечують обслуговування вимог. Кожен канал може обслуговувати не більше ніж одну заявку за раз. Час, необхідний для обслуговування, частіше буває випадковим із певним розподілом імовірності. Кількість каналів обслуговування є обмеженою та визначає її пропускну здатність. Має вплив на час перебування вимог у системі та на ймовірність відмови;

– вихідний потік вимог – це потік вимог, що залишають систему, отримавши чи не отримавши замовлену послугу. Результатом оброблення вимоги є один із ключових критеріїв оцінювання ефективності СМО [2].

Важливим чинником у програмному моделюванні систем масового обслуговування є те, якою мірою механізми мови програмування відповідають критеріям математичного апарату. Одна з основних розбіжностей між мовами програмування, що впливає на доцільність використання для моделювання СМО, полягає в підході до паралельного виконання. Будемо розглядати тільки популярні мови програмування, адже важливу роль відіграє можливість розвивати та підтримувати програмний продукт у довгостроковій перспективі. Відповідно до рейтингу, побудованого способом аналізу вакансій на роль програмного інженера за 2023 р., у табл. 1 наведено десять мов, придатних для написання серверів, що мають долю понад 1%.

Таблиця 1. *Класифікація популярних мов програмування*

Назва	Доля	Підхід до багатопотоковості
JavaScript / TypeScript	29.8%	асинхронність
Python	19.64%	багатопотоковість
Java	17.78%	багатопотоковість + бібліотека для підтримки конкаренсі
C#	12.21%	багатопотоковість
PHP	9.39%	асинхронність
C / C++	9.14%	багатопотоковість + бібліотека для підтримки конкаренсі
Ruby	4.37%	асинхронність
Go	1.91%	SCP, конкаренсі на рівні ядра мови

Серед наведених технологій можна виокремити кілька груп:

– підтримка асинхронності в межах одного потоку операційної системи;

– підтримка роботи з потоками операційної системи напряму;

– підтримка конкаренсі за допомогою розширень, доданих нещодавно як стороння бібліотека;

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №2 (28)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

– повна підтримка конкаренсі та інтеграція ідей *CSP* на рівні ядра мови.

Мови програмування, що донедавна використовуються для написання програм, зокрема й для моделювання систем масового обслуговування, не мали підтримки *CSP* та конкаренсі. Зараз упроваджується часткова підтримка, яка не надає повноти функціоналу [3]. Така ситуація дає змогу застосувати нові підходи до програмного моделювання процесів із використанням сучасних мов програмування.

Мета роботи

Метою статті є обґрунтування доцільності впровадження сучасних комп'ютерних інформаційних технологій, а саме – підходу *CSP* із використанням мови програмування *Go* для моделювання систем масового обслуговування. Також передбачено проаналізувати спорідненість інструментарію паралельного виконання зазначеної технології до математичних узагальнень. Це надалі може відкрити шлях до значного вдосконалення інструментарію та покращення ефективності програмних моделей.

Постановка завдання

Розглядатимемо класичну задачу про багатоканальну систему масового обслуговування з обмеженою чергою вимог та відмовою в разі переповнення черги [1], де n – кількість однакових каналів обслуговування, до яких надходить пуассонівський потік заявок інтенсивності λ. Якщо на момент надходження нової заявки є хоча б один вільний канал, він негайно починає оброблення. Якщо всі канали зайняті – вимога стає останньою до загальної черги ємності k. Заявки покидають чергу для подальшого обслуговування в тій самій послідовності, у якій вони надходили на очікування. Канал, що звільнюється від виконання, відразу починає оброблення першої в черзі вимоги. У цьому разі кожна заявка обслуговується тільки одним каналом і кожен канал може обслуговувати не більше ніж одна вимога одночасно. Якщо вільними є декілька каналів обслуговування, для оброблення буде обрано канал випадковим чином. Час, необхідний на оброблення однієї вимоги, є випадковою величиною з експоненційним законом розподілу ймовірності:

$$F(x) = 1 - e^{-\nu x}, \text{ де } \nu > 0. \qquad (1)$$

Причинами такого рішення є відносна простота самого процесу, наявність потужного математичного апарату аналітичного дослідження [4], а також значна поширеність процесів, що підпадають під цю модель. Отже, задана система підпадає під умовне визначення: $M/M/n/m$, де

– перша M вказує на марковський вхідний потік вимог;

– друга M вказує на те, що процес оброблення вимог також є марковським;

– n визначає, що система є багатоканальною та задає кількість каналів;

– m описує обмежену ємність системи та визначає кількість місць для очікування $m > 0$.

Вхідний потік вимог має задовольняти такі вимоги:

– стаціонарність потоку;

– відсутність післядії;

– ординарність.

Імовірність надходження нової вимоги до системи за проміжок часу t може бути визначений таким чином:

$$P_i(t) = \frac{(\lambda t)^i}{i!} e^{-\lambda t}, \qquad (2)$$

де λ – це інтенсивність прибуття вимог, що надходять до системи за одиницю часу; i – кількість вимог, наявних у системі, разом із наступною в момент t.

Зважаючи на обмеження щодо кількості вимог, які можуть очікувати на обслуговування в черзі, обчислимо ймовірність відмови у виконанні новій вимозі через переповнення черги. Оскільки майбутнє обслуговування не залежить у контексті теорії ймовірностей від того, що відбувалося до моменту часу t_0 у зв'язку з особливостями ймовірнісного розподілу. Для описаної системи ймовірність відмови дорівнює ймовірності розташування в системі рівно i вимог. Маємо такий вираз [1]:

$$\begin{cases} 1 \le i \le n & P_i = \dfrac{\rho^i n^i}{i!} P_0 \\ n < i \ge n + m & P_i = \dfrac{\rho^i n^n}{n!} P_0 \end{cases}, \qquad (3)$$

$$P_0 = \left[\sum_{i=0}^{n-1} \frac{\rho^i n^i}{i!} + \frac{n^n}{n!} \cdot \frac{\rho^n \left(1 - \rho^{m+1}\right)}{1 - \rho} \right]^{-1}, \qquad (4)$$

де $\rho = \dfrac{\lambda}{n\nu}$ – середня продуктивність системи. Нас найбільш цікавить ситуація, коли $\rho > n$, оскільки

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 2 (28)

саме в цьому разі можемо бачити наповнення черги й відмови через її переповнення [1]. Надалі досліджуватимемо методи зменшення ймовірності відмови черговій вимозі. Отже, ймовірність відмови з причини переповнення черги може бути обчислена за допомогою виразу

$$P_{vidm} = P_{n+m} = \frac{\rho^{n+m} n^n}{n!} P_0, \quad (5)$$

$$P_{vidm} = \frac{\rho^{n+m} n^n}{n!} \cdot \left[\sum_{i=0}^{n-1} \frac{\rho^i n^i}{i!} + \frac{n^n}{n!} \cdot \frac{\rho^n \left(1-\rho^{m+1}\right)}{1-\rho} \right]^{-1}. \quad (6)$$

Наведений вираз означає, що запит отримає відмову щодо обслуговування, якщо всі лінії та місця очікування будуть зайняті.

Розв'язання проблеми

Конкаренсі в математиці та інформатиці визначається як властивість систем, у яких кілька обчислень виконуються одночасно та потенційно взаємодіють один з одним [5]. Термін, запропонований Ентоні Хоаром у межах його роботи з *Communicating Sequential Processes* (*CSP*), став основоположним для розуміння комплексних систем, де багато процесів відбуваються одночасно й асинхронно. *CSP* моделює поведінку системи за допомогою алгебри процесів, даючи змогу описати взаємодію між паралельними процесами через події комунікації.

Розвиток теорії *CSP* сприяв упровадженню поняття "конкаренсі" у високорівневих мовах програмування та архітектурах систем. Поняття паралелізму в *CSP* та його математична модель дозволяють проєктувати та аналізувати складні системи, що містять паралельні процеси та взаємодію за допомогою обміну повідомленнями. Теорія конкаренсі застосовується в багатьох галузях разом із розробленням операційних систем, розподіленими обчисленнями та проєктуванням мікросхем. Вона допомагає забезпечити взаємну незалежність, синхронізацію та уникнення взаємоблокувань між паралельними процесами. Окреслений аспект математики та комп'ютерних наук постійно розвивається, оскільки нові парадигми та методології надходять для кращого розуміння й ефективнішого використання паралелізму в обчислювальних системах [6].

Розглянемо застосування підходів конкаренсі до розв'язання поставленого завдання. За термінологією *CSP* багатоканальна система масового обслуговування з обмеженою чергою та відмовами може бути подана у вигляді мережі процесів, що комунікують між собою. Кожен етап, або стан, у якому перебуватиме заявка під час оброблення, є процесом. Перехід між станами відбуватиметься під впливом відповідних подій або сигналів. У цьому разі, згідно із визначенням конкаренсі, усі ці процеси можуть відбуватися асинхронно для кількох заявок водночас. Для початку виокремимо незалежні процеси, що супроводжують оброблення заявки в СМО:

– надходження заявок може мати різні розподілення ймовірностей та інтенсивність;

– утримання в черзі може мати додатковий функціонал визначення пріоритетів;

– оброблення каналом обслуговування може мати випадкову тривалість, а також імовірність помилки;

– покидання заявкою системи, що не залежить від результату обслуговування.

Формалізуємо взаємодію між зазначеними процесами через обмін повідомленнями за допомогою подій, які описуватимуть переходи між станами заявки:

– надходження (*arrive*): подія, що означає надходження чергової заявки до системи та є початковою подією;

– взяття до черги (*onqueue*): за умови, якщо в черзі є вільні місця, заявка успішно стає на очікування;

– покидання черги (*dequeuer*): у разі звільнення хоча б одного каналу обслуговування заявка, що стоїть наступною в черзі, може покидати чергу, звільнюючи місце для подальших надходжень;

– початок оброблення (*start*): канал розпочинає обслуговування заявки;

– успішне завершення оброблення (*success*): опрацювання заявки пройшло в штатному режимі;

– помилка під час оброблення (*fail*): з певних причин канал не зміг надати очікувану послугу;

– відмова (*reject*): у черзі на момент надходження не було вільних місць;

– базова подія завершення оброблення для кінцевих алгоритмів (*STOP*).

Користуючись визначеною множиною подій, сформулюємо алфавіт процесу:

$$aSMO = \{arrive, onqueue, dequeue, start, success, fail, reject\}. \quad (7)$$

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №2 (28)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Тоді процес оброблення заявки матиме такий вигляд [7]:

$$SMO = arrive \to \left(reject \to STOP \middle| onqueue \to dequeue \to start \to \left(success \to STOP \middle| fail \to STOP \right) \right). \quad (8)$$

Отже, бачимо, що системи масового обслуговування, які розглядаються в межах дослідження, можуть бути цілісно описані за допомогою *CSP* та конкаренсі. Крім цього, завдяки абстракціям, а також комунікації через події, ми маємо змогу гнучко масштабувати та розширювати взаємодію компонентів без необхідності змінювати самі процеси. Цей процес може бути запущено паралельно, що не змінить його загальну будову.

Go – це сучасна компільована мова програмування із жорсткою системою типів, що була створена корпорацією *Google* 2012 р. для задоволення нагальних потреб і як відповідь на виклики у сфері комп'ютерної інженерії, що постали перед розробниками програмного продукту в XXI ст., а саме:

– неперервне надходження значної кількості одночасних запитів із подальшим паралельним їх обробленням;

– необхідність підтримки великої різноманітності процесорів, платформ та операційних систем;

– потреба швидко реалізовувати бізнес-ідеї та раніше отримувати відгук про відповідність очікувань;

– бажання спростити написання, розширення, підтримку, а також читання коду програм;

– ідея децентралізованої спільноти розробників і популяризації пакетів із відкритим кодом.

Одним із ключових етапів у формуванні мови *Go* є реалізація ідеї конкаренсі [8]. За цим терміном ховається кілька особливостей рантайму, а саме:

– розбиття складного процесу на ланцюг незначних послідовних завдань;

– взаємодія між процесами відбувається з допомогою сигналів;

– заблоковані завдання стають на очікування, та виконання переходить до інших процесів у черзі;

– здатність легко збільшити кількість обробників для простих завдань.

Кожна із згаданих вище властивостей позначається в теорії масового обслуговування. Тож розглянемо всі особливості підходу та їх вплив на моделювання систем масового обслуговування.

Більшість систем у комп'ютерній інженерії або є дуже складними із самого початку, або стають такими з часом. Їх спрощення є непростим завданням, яке потребує значних зусиль, що пізніше – то більше. Одним з ефективних способів запобігання ускладненню систем є декомпозиція складних завдань на низку простих, маленьких операцій, що взаємодіють між собою. Водночас таке розбиття дає змогу більш ефективно будувати математичні моделі систем, оскільки, замість одного надскладного завдання, будемо моделювати незначні, значно простіші процеси.

Спосіб взаємодії між процесами є дуже важливим, адже самі по собі вони не ефективні. Існує декілька способів комунікації:

– спільний доступ до пам'яті (м'ютекси);

– неблокувальні алгоритми;

– обмін повідомленнями крізь канали.

З-поміж зазначених підходів саме обмін повідомленнями за допомогою каналів найкраще вписується в контекст систем масового обслуговування. Оскільки в цьому підході, на відміну від інших, наголошується не на доступі до даних, а на взаємодії між компонентами [9]. Як наслідок, від проблеми оброблення даних переходимо до питання моделювання взаємодії та визначення станів компонентів системи, що добре описується математичним апаратом.

Хоча *Go* є молодою мовою та має запозичення ідей з інших, старших мов програмування, вона має і свої унікальні властивості, що роблять програми, написані на *Go* ефективнішими та відмінними за характером від програм, написаних спорідненими мовами. Це єдина сучасна технологія, що реалізує принципи конкаренсі, описані в роботах Ч. Е. Р. Хоара "CSP" на рівні ядра [10]. Тому прямий переклад з таких мов, як, наприклад, *Java* або *C++*, малоймовірно дасть очікуваний результат. Відповідно, для ефективного використання мови програмування *Go* важливо розуміти її особливості, ідіоми, а також практики, що дозволяють розкрити потенціал. У *Go* чітко сформульовані стандарти щодо форматування, найменування, будови програми, які дають змогу уніфікувати рішення, а також зробити їх більш зрозумілими для інших розробників і стійкими до змін.

Також ця технологія перебуває в активній фазі розроблення та вдосконалення, і до неї регулярно надходять оновлення, надаючи додаткові можливості та переваги.

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 2 (28)

Ядром програми є її рантайм – набір процесів, що відповідають за оброблення та виконання алгоритму програми, а також забезпечення допоміжних операцій, що відбуваються за замовчуванням на рівні мови [11]. Рантайм у мові *Go* має кілька особливостей, що виділяють її на фоні інших, а саме:

– чітка типізація з нещодавно доданою підтримкою параметризованих типів;

– високий рівень абстракції взаємодії з операційною системою;

– збірник сміття, що працює за методологією *mark & sweep*;

– власний менеджер потоків із вродженою підтримкою конкаренсі.

Ці особливості будуть описані більш детально надалі.

Інструментарій мови програмування *Go* дозволяє дуже гнучко відтворювати різноманітні навантажені системи, багато в чому завдяки спорідненості ідей, закладених у філософію мови, з теорією масового обслуговування [12]. Розглянемо, яким чином можна програмно змоделювати процес масового обслуговування, описаний раніше, за допомогою мови *Go*. Будемо рухатися послідовно, відповідно до життєвого циклу заявки в системі.

Спочатку визначимося з тим, як відтворити вхідний потік вимог. Для цього можемо скористатися функцією, що буде викликатися в циклі. Її відповідальність полягатиме в надсиланні вхідних параметрів далі в чергу для подальшого оброблення. Ця функція буде єдиним публічним контрактом нашої програми, і саме з її допомогою відбуватиметься подання вхідних даних. У користувача програми не буде взаємодії з подальшими етапами.

Наступним кроком буде передача новоствореної вимоги до черги очікування. Для стандартизації логіки виконання ми відмовляємось від прямої комунікації з каналами оброблення вимог і завжди передаватимемо їм роботу через чергу. Такий крок дасть змогу задовольнити вимоги підходу конкаренсі. Це зі свого боку підвищить стійкість програми до помилок, а також спростить її будову й подальшу підтримку. Роль черги в мові програмування *Go* відіграватиме канал з буфером, що дорівнює розміру черги. У разі спроби записати значення до каналу із заповненим буфером рантайм блокуватиме виконання функції. Отже, за необхідності моделювати систему з відмовами скористаємося можливістю додати дію за замовчуванням і повертатимемо помилку переповненої черги, що буде відповідати відмові у виконанні вимоги. Сам по собі буфер каналу в мові програмування *Go* має політику оброблення елементів *FIFO*. Це означає, що вимоги обслуговуватимуться в тій самій послідовності, у якій вони надходили. У разі успішного розташування вимоги в черзі вона має дочекатися, коли всі вимоги, що надійшли раніше, будуть розібрані. Якщо на момент готовності до виконання в наявності понад одна вільна горутина, то вимога буде взята однією з них випадковим чином.

Роль каналу обслуговування в мові *Go* відіграватиме горутина – паралельний процес, що виконує вказану функцію. Вхідні дані для виконання операції горутина вичитує з каналу. Оброблення вхідної інформації відбувається штатним чином і може привести як до досягнення успішного результату, так і спричинити помилку. Коли оброблення вимоги завершене, цикл горутини повертається до спроби читати з каналу та блокується до надходження чергового завдання.

У цьому разі розмір каналу, кількість горутин, функція, що виконуватиме розрахунок, а також структура даних для вимоги задаються на етапі компіляції й не можуть бути змінені в процесі роботи програми. Організувати необхідний режим синхронізації між горутинами та каналами дозволяє директива *select*, роль якої полягає в оркестрації. Час виконання вимоги залежить від функції, заданої для виконання в горутинах, й інженер має змогу визначити її таким чином, щоб задовольнити вимогу до експоненційного випадкового розподілу.

Отже, можемо розглянути базову реалізацію програмного забезпечення для моделювання СМО з обмеженою чергою, без відмов, написану мовою *Go*. Першою компонентою буде функція з назвою *feed*, завданням якої є лише подання заявок до системи масового обслуговування (рис. 1).

```
func feed(inCh chan<- inputData) {  1 usage
    for i := 0; i < requestsNumber; i++ {
        inCh <- inputData{id: i, duration: time.Millisecond * time.Duration(i%3)}
    }
    close(inCh)
}
```

Рис. 1. Функція подання заявок

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №2 (28)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

Коли всі запити надіслано до черги, закриваємо канал заявок, сигналізуючи іншим компонентам про цю подію. Другою операцією розберемо функцію *service*, що описує поведінку каналу обслуговування та виконує примітивну логіку: повернути помилку, якщо значення *id* рівняється по модулю 3, а інакше – повернути успіх (рис. 2). Функція *newSMO* запускає канали обслуговування та відповідальна за закриття каналу відповідей після завершення оброблення даних (рис. 3). Усі раніше згадані операції будуть викликані у функції *main* у момент запуску програми (рис. 4), також у ній читаємо результати оброблення заявок з каналу й виводимо їх у командний рядок, продовжуючи ці дії до закриття каналу.

```
func service(inCh <-chan inputData, outCh chan<- outputData, wg *sync.WaitGroup) {  1 usage
    for in := range inCh {
        if in.duration == 0 {
            outCh <- outputData{id: in.id, err: errors.New( text: "service duration can not be 0")}
            continue
        }

        time.Sleep(in.duration)
        outCh <- outputData{id: in.id}
    }

    wg.Done()
}
```

Рис. 2. Функція обслуговування

```
func newSMO(numberOfServices int, inCh <-chan inputData, outCh chan<- outputData) {  1 usage
    wg := sync.WaitGroup{}
    wg.Add(numberOfServices)
    for i := 0; i < numberOfServices; i++ {
        go service(inCh, outCh, &wg)
    }

    wg.Wait()
    close(outCh)
}
```

Рис. 3. Функція ініціалізації СМО

```
func main() {
    inCh := make(chan inputData, numberOfServices)
    outCh := make(chan outputData, queueSize)

    go feed(inCh)
    go newSMO(numberOfServices, inCh, outCh)

    for res := range outCh {
        if res.err != nil {
            log.Printf( format: "Service for the inpput id=%d failed because: %v\n", res.id, res.err)
            continue
        }
        log.Printf( format: "Service for the inpput id=%d successfuly finshed", res.id)
    }
}
```

Рис. 4. Основна функція виклику та оброблення результатів

Результати дослідження

Мова програмування *Go* перезавантажує підхід до паралельного виконання програм унаслідок інтеграції методології конкаренсі у філософію та інструментарій мови. Вона є першопрохідницею, оскільки саме в *Go* вперше було реалізовано концепцію *CSP* на рівні базових механізмів.

Горутини дуже добре відтворюють поведінку незалежних каналів оброблення вимог, оскільки так само не поділяють між собою стан і не впливають напряму на взаємне виконання вимог. За потреби

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) Innovative technologies and scientific solutions for industries. 2024. No. 2 (28)

можна вказати кількість горутин як будь-яке ціле додатне число. Однак важливо не забувати, що навіть дуже легкі потоки можуть стати проблемою за умови відсутності контролю за їх виконанням та вчасним коректним завершенням. Відповідно, необхідно наслідувати певні практики для зменшення ймовірності неочікуваної поведінки, а також застосовувати ефективні методики для виявлення, пошуку джерела та усунення помилок, пов'язаних із конкаренсі в програмі [13].

Канал у *Go* повністю відповідає ролі черги, він надає послідовність очікування вимог, синхронізацію доступу, а також випадковість підбору горутини для виконання. Якщо потрібно, можна прибрати буфер у каналу, що призведе до трансформації на систему з відмовами та без черги. У разі необхідності змоделювати систему без відмов у інструментарії розробника є слайс, що є масивом із здатністю автоматично збільшувати свою ємність. Отже, він може відігравати роль умовно необмеженої черги.

Функції, що виконуватимуть розрахунки, є дуже гнучким механізмом і дозволяють задовольнити будь-які вимоги до часу розв'язання задачі залежно від потреб експерименту.

Однією з проблем, що постає під час дизайну мови програмування, є задача про багатопотоковість. Оскільки логіка виконуватиметься на потоках операційної системи, популярним підходом є пряме поєднання програмних потоків із потоками ОС. Серед переваг такого рішення – його відносна простота. Однак програми стають залежними від того, яким чином операційна система виділяє їм процесорний час. У мові *Go* розробники взяли за основу ідею про легкі, маленькі програмні потоки, які не мають прямого виходу на рівень операційної системи [14]. Значною особливістю мови програмування *Go* є її скедулер (планувальник) – механізм аркестрації горутин. Система планування багатопотоковості має такі компоненти:

– потік ядра процесора (*Core*);

– потік операційної системи (*M*), що взаємодіє з потоком процесора;

– програмний потік (*P*) – абстракція рівня рантайму *Go*, що зазвичай відповідає одному потоку операційної системи;

– горутина (*G*) – маленький автономний процес, що описує бізнес-логіку;

– допоміжні процеси рантайму – операції, що є горутинами самі по собі й забезпечують життєдіяльність програми;

– скедулер (планувальник) – алгоритм, що обирає, коли та якій горутині дати змогу виконання, а також який тред (потік) операційної системи її обслуговуватиме.

Опишемо роботу механізму взаємодії компонентів планувальника (рис. 5).

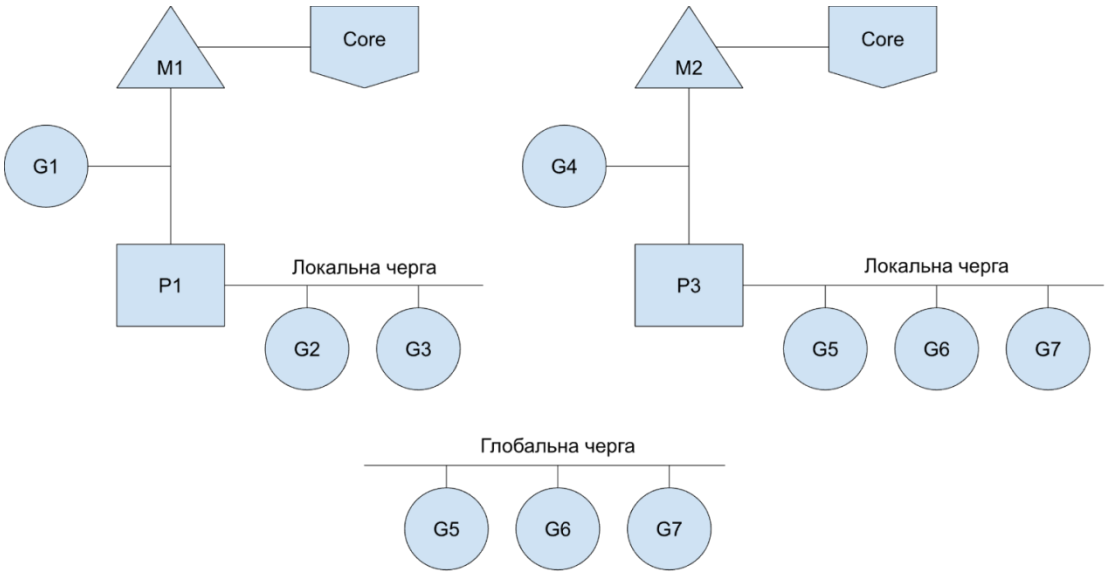

Рис. 5. Компонентна схема планувальника

1. У процесі використання в програмному коді директиви *Go* із подальшим визначенням, яка функція має виконуватись, рантайм створює маленьку горутину з власним стеком пам'яті мінімального розміру.

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №2 (28)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

2. Планувальник визначає локальну чергу програмного потоку, до якої потрапить горутина на очікування.

3. Коли потік буде готовий взяти на виконання чергову горутину, він бере її в роботу.

4. Є перелік умов, за яких виконання горутини може бути тимчасово призупинене з міркувань надання часу іншим горутинам. У цьому разі горутина, що була попередньо в обробленні, "паркується" до наступного вікна можливостей.

5. У разі закінчення очікувальних горутин у локальній черзі програмного потоку він виконає спробу забору половини черги горутин у іншого, більш завантаженого програмного потоку.

6. Раз за певний час або якщо всі локальні черги пусті, забір відбувається з глобальної черги.

Основна ідея такого підходу – надання додаткового рівня абстракції для щільнішого планування роботи потоків операційної системи. Цей механізм забезпечує дію підходу конкаренсі, навіть за наявності лише одного ядра процесора [15].

Найважливішою перевагою в сукупності факторів є загальна простота взаємодії елементів, масштабування та потенціал до паралельного виконання на багатоядерних процесорах [16]. В аналогічний спосіб можна розширювати можливості [17].

Висновки

Мова програмування *Go* – дуже вдала технологія для моделювання систем масового обслуговування. Її філософія, спосіб роботи, а також вбудований інструментарій має широкі можливості для моделювання різноманітних систем масового обслуговування. Методологія конкаренсі, що була взята за основу ядра *Go*, є основною відмінністю від інших мов програмування. Запропонована та описана в межах наукової праці Ч. Е. Р. Хоара "CSP", вона описує новий підхід до побудови абстракцій навколо взаємодії асинхронних процесів. Це дуже вдало розширює інструментарій інженера механізмами взаємодії складників системи. У цьому разі виконання програмної реалізації моделі значною мірою наближається до досліджуваного процесу. Черговою перевагою застосування *Go*, порівняно з іншими поширеними мовами програмування, є її підвищена швидкодія. Важливу особливість також становить простота у використанні інструментарію та подальша підтримка готового продукту, що особливо помітно в роботі з багатопотоковістю.

Отже, використання мови програмування *Go* є доцільним і дає змогу наблизити поведінку програми до модельованого процесу, спростити імплементацію та зменшити час, необхідний для оброблення даних.

Наукова новизна

Набув подальшого розвитку інструментарій для комп’ютерного моделювання систем масового обслуговування. Запропоновано застосування підходів конкаренсі, їх імплементації в мові програмування *Go* до моделювання систем масового обслуговування. Це матиме такі наслідки:

– підвищить ефективність розрахунків;

– спростить програмну реалізацію;

– наблизить програмну реалізацію до модельованого процесу.

Перспективи

Окреслений напрям розвитку методів програмного моделювання систем масового обслуговування має значний потенціал. Надалі необхідно спроєктувати та реалізувати програмний продукт мовою *Go*, що даватиме змогу моделювати різноманітні процеси. На базі такого програмного рішення буде можливо перевіряти на практиці різноманітні гіпотези щодо оптимізації в системах масового обслуговування. Ключовою вимогою для цього завдання буде універсальність і забезпечення можливостей щодо моделювання широкого спектра систем масового обслуговування.

Список літератури

1. Литвинов А.Л. Теорія систем масового обслуговування. Харків: ХНУМГ ім. О.М. Бекетова, 2018. 141 с.
2. Borovkov A. Stochastic Processes in Queueing Theory. Translate by K.Wickwire. *New York: Springer-Verlag*, 1976. 280 p. DOI: https://doi.org/10.1007/978-1-4612-9866-3

ISSN 2522-9818 (print)
ISSN 2524-2296 (online) *Innovative technologies and scientific solutions for industries. 2024. No. 2 (28)*

3. Chabbi M., Ramanathan M.K. A study of real-world data races in Golang. *PLDI 2022: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, San Diego, 13–17 June 2022 / Special Inter. Group on Program. Lang. SIGPLAN. New York, 2022. P. 474–489. DOI: https://doi.org/10.1145/3519939.3523720
4. Zeifman A.I. Quasi-Markov Processes and Description of Some Models of Queueing Theory. *IFAC Proceedings Volumes.* 1986. Vol. 19. No. 5. P. 445–449. DOI: https://doi.org/10.1016/S1474-6670(17)59840-7
5. Hoare A. Communicating sequential processes. New Jersey: Prentice Hall, 1985. 235 p. DOI: https://doi.org/10.1145/359576.359585
6. Yunfei Liu EMsFEM based concurrent topology optimization method for hierarchical structure with multiple substructures/ Yunfei Liu et al. *Computer Methods in Applied Mechanics and Engineering.* 2024. Vol. 418. Part A. 116549. P. 1–26. DOI: https://doi.org/10.1016/j.cma.2023.116549
7. C. A. R. Hoare. Communicating sequential processes. *Communications of the ACM*. 1978. Vol. 21. No 8. P. 666–677. DOI: https://doi.org/10.1145/359576.359585
8. Sufyan bin U. Mastering GoLang: A Beginner's Guide. Boca Raton: CRC Press, 2022. 298 p. DOI: https://doi.org/10.1201/9781003310457
9. Fava D., Steffen M. Ready, set, Go!: Data-race detection and the Go language. *Science of Computer Programming.* 2020. Vol. 195. 102473. P.1–23. DOI: https://doi.org/10.1016/j.scico.2020.102473
10. Sottile M., Mattson T., Rasmussen C. Introduction to Concurrency in Programming Languages. New York: Chapman and Hall/CRC, 2009. 344 p. DOI: https://doi.org/10.1201/b17174
11. Sufyan bin U. GoLang: The Ultimate Guide. Boca Raton: CRC Press, 2022. 366 p. DOI: https://doi.org/10.1201/9781003309055
12. Bowman H., Gomez R. Concurrency Theory: Calculi an Automata for Modelling Untimed and Timed Concurrent Systems. London: Springer-Verlag, 2006. 422 p. DOI: https://doi.org/10.1007/1-84628-336-1
13. Zhang D., Qi P., Zhang Y. GoDetector: Detecting Concurrent Bug in Go. *IEEE Access.* 2021. Vol. 9. P. 136302–136312. DOI: https://doi.org/10.1109/ACCESS.2021.3116027
14. Donovan A. A. A., Kernighan B. W. The Go programming language. New York: Addison-Wesley Professional, 2015. 400 p.
15. Pontelli E., Gupta G. On the duality between or-parallelism and and-parallelism in logic programming. *1st International EURO-PAR Conference on Parallel Processing:* EURO-PAR 199. Stockholm, 29–31 aug. 1995. Lecture Notes in Computer Science. Vol 966. Springer, Berlin. 2005. P. 43–54. DOI: https://doi.org/10.1007/BFb0020454
16. Komendantskaya E., Schmidt M., Heras J. Exploiting Parallelism in Coalgebraic Logic Programming. *Electronic Notes in Theoretical Computer Science.* 2014. Vol. 303. P. 121–148. DOI: https://doi.org/10.1016/j.entcs.2014.02.007
17. Whitney J., Gifford C., Pantoja M. Distributed execution of communicating sequential process-style concurrency: Golang case study. *The Journal of Supercomputing.* 2019. Vol. 75. No 3. P. 1396–1409. DOI: https://doi.org/10.1007/s11227-018-2649-2

References

1. Lytvynov, A. (2018), *Queuing Theory System, [Teoriia system masovoho obsluhovuvannia]*, KhNUMH im. O.M. Beketova, Kharkiv, 141 p.
2. Borovkov, A. (1976), *Stochastic Processes in Queueing Theory. Translate by K.Wickwire*, Springer-Verlag, New York, 280 p. DOI: https://doi.org/10.1007/978-1-4612-9866-3
3. Chabbi, M., Ramanathan, M. (2022), "A study of real-world data races in Golang", *PLDI 2022: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 13–17 June 2022, San Diego. Special Inter. Group on Program. Lang. SIGPLAN, New York, P. 474–489. DOI: https://doi.org/10.1145/3519939.3523720
4. Zeifman, A. (1986), "Quasi-Markov Processes and Description of Some Models of Queueing Theory", *IFAC Proceedings Volumes*, Vol. 19, No. 5, P. 445–449. DOI: https://doi.org/10.1016/S1474-6670(17)59840-7
5. Hoare, A. (1985), *Communicating sequential processes*, Prentice Hall, New Jersey. 235 p. DOI: https://doi.org/10.1145/359576.359585
6. Liu, Y. (2024), "EMsFEM based concurrent topology optimization method for hierarchical structure with multiple substructures"/ Liu, Y. et al., *Computer Methods in Applied Mechanics and Engineering*, Vol. 418, Part A, 116549, P. 1–26. DOI: https://doi.org/10.1016/j.cma.2023.116549
7. Hoare, A. (1978), "Communicating sequential processes", *Communications of the ACM*, Vol. 21, No 8, P. 666–677. DOI: https://doi.org/10.1145/359576.359585
8. Sufyan, bin U. (2022), *Mastering GoLang: A Beginner's Guide*, CRC Press, Boca Raton, 298 p. DOI: https://doi.org/10.1201/9781003310457
9. Fava, D., Steffen, M. (2020), "Ready, set, Go!: Data-race detection and the Go language", *Science of Computer Programming*, Vol. 195, 102473, P.1–23. DOI: https://doi.org/10.1016/j.scico.2020.102473
10. Sottile, M., Mattson, T., Rasmussen, C. (2009), *Introduction to Concurrency in Programming Languages*, Chapman and Hall/CRC, New York, 344 p. DOI: https://doi.org/10.1201/b17174
11. Sufyan, bin U. (2022), *GoLang: The Ultimate Guide*, CRC Press, Boca Raton, 366 p. DOI: https://doi.org/10.1201/9781003309055

Сучасний стан наукових досліджень та технологій в промисловості. 2024. №2 (28)
ISSN 2522-9818 (print)
ISSN 2524-2296 (online)

12. Bowman, H., Gomez, R. (2006), *Concurrency Theory: Calculi an Automata for Modelling Untimed and Timed Concurrent Systems*, Springer-Verlag, London, 422 p. DOI: https://doi.org/10.1007/1-84628-336-1
13. Zhang, D., Qi, P., Zhang, Y. (2021), "GoDetector: Detecting Concurrent Bug in Go", *IEEE Access*, Vol. 9, P. 136302–136312. DOI: https://doi.org/10.1109/ACCESS.2021.3116027
14. Donovan, A., Kernighan, B. (2015), *The Go programming language*, Addison-Wesley Professional, New York, 400 p.
15. Pontelli, E., Gupta, G. (2005), "On the duality between or-parallelism and and-parallelism in logic programming", *1st International EURO-PAR Conference on Parallel Processing: EURO-PAR 199*, Stockholm, 29–31 aug. 1995. Lecture Notes in Computer Science, vol 966, Springer, Berlin, P. 43-54. DOI: https://doi.org/10.1007/BFb0020454
16. Komendantskaya, E., Schmidt, M., Heras, J. (2014), "Exploiting Parallelism in Coalgebraic Logic Programming", *Electronic Notes in Theoretical Computer Science*, Vol. 303, P. 121–148. DOI: https://doi.org/10.1016/j.entcs.2014.02.007
17. Whitney, J., Gifford, C., Pantoja, M. (2019), "Distributed execution of communicating sequential process-style concurrency: Golang case study", *The Journal of Supercomputing*, Vol. 75, No 3, P. 1396–1409. DOI: https://doi.org/10.1007/s11227-018-2649-2

Надійшла (Received) 09.05.2024

Відомості про авторів / About the Authors

Гольдінер Денис Ігорович – Харківський національний університет радіоелектроніки, аспірант, Харків, Україна; e-mail: denys.holdiner@nure.ua; ORCID ID: https://orcid.org/0000-0002-1456-1867

Goldiner Denys – Kharkiv National University of Radio Electronics, PhD Student, Kharkiv, Ukraine.

APPLICATION OF GO PROGRAMMING LANGUAGE FOR SIMULATION OF MASS SERVICE PROCESSES

The **subject matter** of the article is methods and approaches to software modeling of System of Mass Services on the example of a multi-channel system with a limited queue and failures in case of its overflow. The **goal** of the work is to justify the feasibility of using modern computer information technologies, namely the Go programming language for modeling System of Mass Services. The following **tasks** were solved in the article: formulation of the researched System of Mass Services; determination of components, criteria of kinship of System of Mass Services with their software models; general overview of concurrency as a mathematical model; a description of the approaches and tools of the Go programming language. The following **methods** are used: Go programming language and its tools, concurrency, parallel execution. The following **results** were obtained: the researched task of mass service was formulated; criteria for comparing the components of System of Mass Services with the Go programming language toolkit were formed; an analysis of the feasibility of using the Go programming language for modeling System of Mass Services was carried out; received further development of tools for computer simulation of System of Mass Services; the application of concurrency approaches, their implementation in the Go programming language, to the modeling of System of Mass Services is proposed. **Conclusions**: The Go programming language is a very successful technology for modeling System of Mass Services. Its philosophy, way of working, as well as the built-in toolset provide ample opportunities for modeling various System of Mass Services. The use of this language is appropriate and allows to bring the behavior of the program closer to the simulated process, simplify implementation, and reduce the time required for data processing. There are great prospects for the further implementation of a software product implemented in the Go language in the field of mass service process modeling.

Keywords: computer simulation; System of Mass Services; Go programming language; concurrency.

Бібліографічні описи / Bibliographic descriptions

Гольдінер Д. І. Застосування мови програмування *GO* для моделювання процесів масового обслуговування. *Сучасний стан наукових досліджень та технологій в промисловості*. 2024. № 2 (28). С. 65–75. DOI: https://doi.org/10.30837/2522-9818.2024.2.065

Goldiner, D. (2024), "Application of GO programming language for simulation of mass service processes", *Innovative Technologies and Scientific Solutions for Industries*, No. 2 (28), P. 65–75. DOI: https://doi.org/10.30837/2522-9818.2024.2.065