УДК 681.03

ВИКОРИСТАННЯ МОДЕЛІ АКТОРІВ ДЛЯ РЕАЛІЗАЦІЇ РОЗПОДІЛЕНИХ ГЕНЕТИЧНИХ АЛГОРИТМІВ

М.М. ГЛИБОВЕЦЬ, С.О. ЗІНЧУК

Досліджено можливості застосування моделі акторів як засобу проектування та аналізу розподілених програмних систем з високою завантаженістю. Основну увагу приділено використанню моделі акторів для реалізації паралельного розподіленого генетичного алгоритму. Здійснено огляд різноманітних моделей паралельних розподілених генетичних алгоритмів, окреслено їхні переваги та недоліки. Для концепції «господар-робітники» запропоновано адаптацію її синхронного та асинхронного варіантів до моделі акторів. Засобами фреймворку Akka створено розподілену систему — кластер акторів. У середовищі кластера описано розгортання застосунка, який демонструє використання пропонованої адаптації концепції «господар-робітники» для розв'язання задачі пошуку найкращої стратегії поведінки робота у штучному середовищі.

ВСТУП

Проблемам створення масштабованих застосувань, здатних одночасно оброблювати великі обсяги даних, завжди приділялася значна увага. Модель акторів (Actor model) як один із підходів до побудови таких систем постала на ґрунті фізичних ідей, зокрема квантової фізики та загальної теорії відносності. Вперше «модель акторів» з'явилась у працях групи дослідників під керівництвом Карла Х'юітта [1], а після публікації у 1986 р. праці Гуля Аги відбулося остаточне становлення моделі [2]. Логічним підсумком досліджень, що відбувалися у Федеральній політехнічній школі Лозанни стала поява фреймворку Akka, який пропонує інструментарій для створення відмовостійких масштабованих розподілених застосунків [3–4]. Наразі частина фреймворку, що відповідає за імплементацію моделі акторів, стала частиною програмної системи мови Scala.

На тлі поширення багатоядерних систем та гібридних архітектур апаратного забезпечення відбувається переосмислення існуючих алгоритмів, їхня адаптація під наявні обчислювальні потужності. Це стосується і генетичних алгоритмів. Особливостям їхньої роботи на багатоядерних та мультиядерних системах присвячено чимало публікацій [5].

Наразі генетичні алгоритми розглядають як стохастичні пошукові алгоритми, що успішно використовуються для розв'язання різноманітних наукових та інженерних задач і допомагають отримувати наближені розв'язки за прийнятний проміжок обчислювального часу [6–8].

Аналіз наукової літератури останніх десятиріч свідчить про інтенсивну розробку підходів до побудови і використання еволюційних та генетичних алгоритмів, зокрема орієнтованих на сучасну архітектуру обчислювальних систем. Огляд наукових досліджень та можливостей існуючих програмних застосунків можна знайти в роботах [9–12].

Мета роботи — дослідження можливостей застосування особливостей моделі акторів як засобу проектування та аналізу розподілених програмних систем з високою завантаженістю, що здатні обробляти великі обсяги даних та ефективно використовувати наявні обчислювальні потужності, для реалізації паралельного розподіленого генетичного алгоритму.

РОЗПОДІЛЕНІ ГЕНЕТИЧНІ АЛГОРИТМИ

Розвиток апаратного забезпечення, зокрема поява багатоядерних та графічних процесорів, створення на основі їхнього поєднання гетерогенних архітектур, спричинив хвилю переосмислення вже існуючих реалізацій різноманітних алгоритмів з метою максимально ефективного використання апаратних ресурсів. Послідовні версії генетичних алгоритмів стикаються з певними проблемами, зумовленими особливостями задач, що розв'язуються. Для деяких задач розмір популяції має бути дуже великим і обсяг пам'яті, який виділяється для зберігання одного індивіда також може бути досить значним, особливо у випадку використання таких нетривіальних структур даних як графи та матриці. Обчислення функції пристосованості для усіх особин популяції за таких умов стає досить затратною операцією з точки зору часових ресурсів. Зрештою послідовні генетичні алгоритми можуть потрапити в пастку субоптимального регіону простору пошуку, що стає на заваді знаходженню більш прийнятних розв'язків. Натомість паралельні розподілені варіанти генетичних алгоритмів можуть здійснювати пошук у різних підпросторах простору пошуку, зменшуючи таким чином імовірність концентрації особин лише у підпросторах з низькою якістю рішень, наприклад, підпросторах локальних мінімумів у випадку оптимізаційних задач.

Основною перевагою таких алгоритмів є більш висока продуктивність у порівнянні з послідовними алгоритмами, навіть у випадку використання не надпотужного апаратного забезпечення. Підставою для цього є те, що кілька популяцій забезпечують процес видоутворення — еволюцію окремих груп особин у різних напрямах (до різних оптимумів). У зв'язку з цим запропоновано вважати окремі розподілені моделі не тільки розширенням традиційних послідовних генетичних алгоритмів, а й новим класом алгоритмів, що по-іншому опрацьовує простір пошуку.

До основних моделей паралельних розподілених генетичних алгоритмів належать: «господар–робітники», дрібнозернисті, грубозернисті та гібридні моделі.

Метод «господар-робітники» належить до найбільш популярних методів, його успішні реалізації з'явились одними з перших [12]. Цей метод ще називають розподіленим обчисленням функції пристосованості. В алгоритмі використовується єдина популяція, якою опікується господар (окремий процесор). Оскільки обчислення функції пристосованості для конкретних індивідів з одного боку є найбільш затратним, а з другого — не потребує ніякої додаткової інформації окрім самого індивіда, господар делегує його підлеглим робітникам. Безпосереднє паралельне обчислення пристосованості здійснюється через присвоєння кожному з наявних процесорів-робітників частини популяції (в ідеальному випадку по одній особині на обчислювальний елемент). Комунікація між робітниками та господарем відбувається лише під час отримання особин для обробки та повернення отриманих значень функції пристосованості. Результати обчислення можуть повертатись у спільну

пам'ять або передаватись за допомогою обміну повідомленнями, що дозволяє господареві зчитувати отримані значення. Вибірка найкращих особин з популяції та схрещування здійснюється глобально, оскільки кожна особина може змагатись та схрещуватись з будь-яким іншим індивідом з популяції. Робітники, окрім обчислення функції пристосованості, можуть також застосовувати оператор мутації та іноді обмінюватись частиною бітів генома (виконувати частину оператора кросинговера).

Модель «господар–робітники» може бути реалізована двома різними способами: синхронним та асинхронним. Алгоритм вважається синхронним, якщо господар, делегуючи обчислення функції пристосованості робітникам, чекає, доки не буде обраховано пристосованість кожної особини у цьому поколінні. Лише після завершення обробки усієї поточної популяції засобами селекції та інших операторів відбувається перехід до формування нового покоління. Натомість в асинхронній версії алгоритму господар, делегувавши обчислення пристосованості кожному робітникові, не чекає на повільні процесори. Формування нового покоління починається тільки після того, як певна частина покоління буде оброблена робітниками. Результати роботи синхронної версії «господар-робітники» не відрізняються від послідовного алгоритму, оскільки концептуальних змін не відбувається. Проте асинхронна версія породжує інші результати, що пов'язано зі змінами в логіці роботи алгоритму. Синхронну версію відносно легко реалізувати і в ідеальному випадку вона може забезпечити значне прискорення, якщо накладні витрати на комунікацію не перевищуватимуть обчислювальних витрат. Зазначимо, що час виконання зменшується майже лінійно зі зростанням кількості процесорів, які задіяні в якості робітників. Однак у цій версії є класичне «вузьке місце» — господар повинен чекати доки найповільніший з процесорів поверне результат обчислення функції пристосованості. Лише після цього можна застосувати оператор селекції. Асинхронна версія долає цей недолік, проте поведінка алгоритму суттєво змінюється. Як зазначено у [12], найдоцільнішим способом реалізації асинхронної моделі «господар–робітники» може бути застосування турнірного відбору серед лише тієї частини популяції, пристосованість якої вже встигли обчислити робітники.

Важливим класом є моделі зі статичними підпопуляціями, що використовують додатковий оператор міграції. Підпопуляції прийнято називати демами. Такі алгоритми ще називають алгоритмами з кількома демами та грубозернистими. У природі загальне видове різноманіття досягається завдяки ізоляції. Схрещування та природний відбір відбуваються окремо для кожної підпопуляції. Це спричинено географічними обмеженнями на кшталт гірських долин та островів, що обмежує мобільність індивідів. Ця концепція кількох окремих демів, які допомогли біоті (флорі та фауні певного району) розвинутись, була запозичена й адаптована до концепції генетичних алгоритмів. Загальна популяція розбивається на деяку кількість демів, що відокремлені один від одного. Індивіди змагаються між собою лише в межах одного дема. Оскільки повна ізоляція демів призвела б до виродження через певну кількість поколінь, вводиться додатковий оператор міграції, що в певні часові проміжки забезпечує обмін індивідами. Якщо особини можуть мігрувати до будь-якої підпопуляції, модель називають «острівною». У випадку міграції лише до сусідніх демів маємо модель крокових камінців. Можуть використовуватись також інші моделі міграції. Добре збалансована міграція

може сприяти поширенню індивідів поміж островами-популяціями і водночас забезпечувати різноманіття окремих островів завдяки підтримці їхньої ізоляції.

У порівнянні з послідовними генетичними алгоритмами, використання підпопуляцій дозволяє більш явно досліджувати пошуковий простір і завдяки загальному різноманіттю видів побороти стагнацію популяції. Однак потрібно підкреслити, що логіка роботи алгоритму, особливо у випадку острівної моделі, дещо змінюється завдяки міграції, забезпечуючи швидшу збіжність. Міграція постає ключовим нетривіальним елементом цієї моделі.

Дрібнозернистий алгоритм споріднений з острівною моделлю щодо розбиття глобальної популяції на окремі менші частини. Проте в цьому випадку розбиття виконується на велику кількість маленьких демів. Звідси виникає потреба у великій кількості процесорів, оскільки в ідеалі кожна підпопуляція має складатись з одного індивіда. Комунікація між демами як і раніше може здійснюватись за допомогою оператора міграції або завдяки використанню демів, які накладаються один на одного. Зазвичай певна просторова структура обмежує взаємодію між підпопуляціями. Дем може змагатись та схрещуватись лише з сусідами, але оскільки окремі популяції є сусідами для кількох інших, кращі розв'язки переміщуються по всій глобальній популяції. У цій моделі тиск селекції та швидкість поширення інформації можуть налаштовуватись завдяки зміні кількості сусідів підпопуляції. Це означає, що дрібнозерниста модель відрізняється від послідовної версії не тільки за часом виконання, але й з точки зору роботи алгоритму. Швидкість поширення інформації також залежить від обраної топології розташування підпопуляцій. Досить поширеною практикою є розташування підпопуляцій у двовимірному гріді. Можливо використовувати архітектуру гіперкубу та інші схеми маршрутизації, а саме: кільце, тор, $16\times16\times8$ куб, $4\times4\times4\times4$ гіперкуб та десятивимірний бінарний гіперкуб. Крім того, зустрічається структура у вигляді островів, в якій лише один індивід з кожного дему перетинається з іншими демами. Підсумовуючи, зазначимо, що цю модель може бути реалізовано на кластері робочих станцій (хоча питання комунікаційних витрат лишається відкритим) та на мультипроцесорі.

Підсилюючи позитивні властивості описаних вище моделей, цей підхід має на меті їх об'єднати та компенсувати недоліки. У [13] пропонується розглядати дворівневу ієрархічну модель, на верхньому рівні якої знаходиться грубозернистий підхід, на нижньому — модель «господар–робітники». Грубозернистий підхід надає нижньому рівню популяцію відповідного розміру та може забезпечувати обмін індивідами з іншими підпопуляціями.

ВІДОБРАЖЕННЯ ГЕНЕТИЧНИХ АЛГОРИТМІВ НА МОДЕЛЬ АКТОРІВ

Використання акторів для вирішення складних предметно-орієнтованих задач потребує впорядкованих систем акторів. В окремих випадках задача вимагає створення групи з кількох систем, кожна з яких виконує прив'язаний суто до неї блок завдань. З'являється потреба в чіткому розподілі ролей серед акторів, створенні ієрархії підпорядкування (керівник–підлеглий). Кілька теоретичних моделей упорядкування систем акторів та синхронізації їхньої діяльності, зокрема адаптація підходу Map-Reduce, представлені у [14].

Однією з найбільш природних моделей для упорядкування взаємодії акторів постає модель «господар–робітники». Застосування цієї моделі для побудови інвертованого індексу, обрахунку числа Пі та інших задач представлені у [15–16]. В адаптації концепції «господар–робітники» до моделі акторів головним постає «актор–господар» (Master Aktor). Він є точкою входу для комунікації з зовнішнім світом (іншими модулями системи), до якої надходять данні для обробки або відбувається запит на отримання результатів роботи. Поряд з господарем в одній системі акторів співіснують а «актори–робітники». Іноді створюються окремі системи акторів суто для робітників. Лише господар приймає рішення щодо розподілу завдань серед підлеглих акторів. Також в окремих випадках він може корегувати кількість робітників, яку потрібно створити, та тривалість їхнього співіснування в системі (господар може примусово зупинити кількох робітників для звільнення ресурсів). Натомість «актори–робітники» лише виконують поставлені для них завдання і повертають отримані результати господареві. Зазвичай задачі для робітників формулюються таким чином, що для їх виконання потрібен мінімальний набір даних і немає потреби у додаткових знаннях про оточуючий світ. «Актор–господар» виступає «мозковим центром» застосунку. На нього покладається відповідальність щодо упорядкування отриманих від підлеглих проміжних результатів і формування на їх основі фінального. Принципову схему моделі «господар–робітники» зображено на рис. 1.

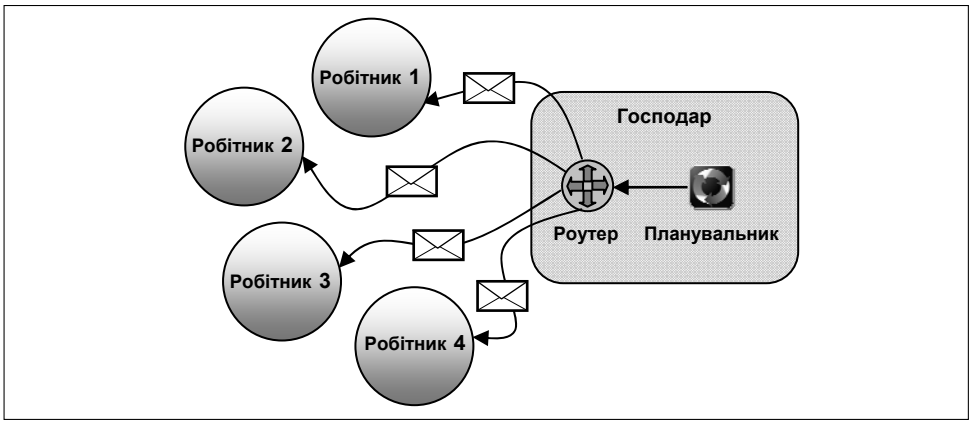

Рис. 1. Принципова схема моделі «господар–робітник» [15]

Адаптація застосування акторів для реалізації розподіленого генетичного алгоритму виглядає наступним чином. «Актор–господар» зберігає у собі популяцію особин. Створивши залежно від особливостей предметної області потрібну кількість акторів-робітників, він віддає кожному з них частину популяції для обчислення функції пристосованості. Кількість особин, яку отримає кожен актор для обробки, на наш погляд, варта окремого дослідження. Залежно від того, який з варіантів моделі обрано, господар може чекати доки всі робітники надішлють йому повідомлення зі значеннями функції пристосованості або встановити певний ліміт на кількість особин, яку потрібно оцінити. Як тільки бажану частину поточної популяції оцінено, господар може перейти до формування нового покоління. «Актори–робітники» також можуть застосувати оператор мутації щодо особин поточної популяції та окремі частини інших операторів (залежить від їх складності). Рішення про те, що найкращий розв'язок знайдено, приймає «актор–

господар». Також саме йому в якості вхідних параметрів передається імовірність кросоверу, мутації, розмір популяції та інші необхідні обмеження. Задля того, щоб розмежувати логіку генетичного алгоритму від контролю життєвого циклу підлеглих акторів, є доречним ввести додаткові сутності: «актор–монітор» та «актор–наглядач». «Актор–монітор» відслідковує те, чи не зупинився випадково якийсь з «акторів–робітників». Наглядач контролює кількість дійсно працюючих підлеглих, відправку до них повідомлень, і за потреби може ініціювати перезапуск відповідного «актора-робітника» [15]. Схему цього розширення представлено на рис. 2.

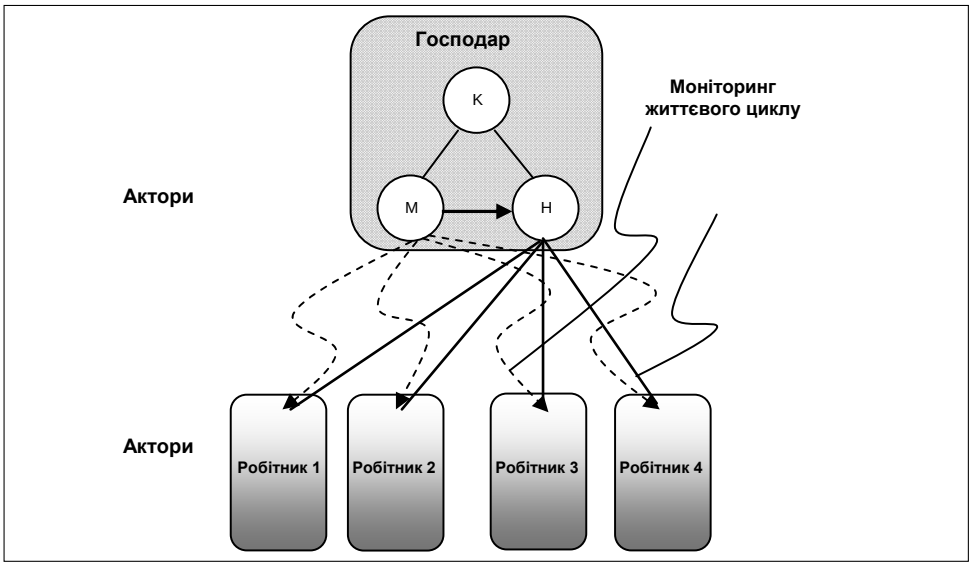

Рис. 2. Додаткові актори для контролю життєвого циклу робітників [16]: M — актор-монітор, H — актор-наглядач, K — актор, який контролює роботу та життєвий цикл M та H

РЕАЛІЗАЦІЯ РОЗПОДІЛЕНОГО ГЕНЕТИЧНОГО АЛГОРИТМУ

Для демонстрації відображення концепції «господар–робітники» на модель акторів було використано задачу, запропоновану в [17]. У ній описується робот Роббі, який існує в уявному світі, наповненому певними ресурсами (в оригіналі порожніми бляшанками від напою). Основна мета робота — зібрати якомога більше ресурсів (очистити світ від сміття). Двовимірний світ Роббі складається зі 100 клітинок, упорядкованих в сітку розмірами 10×10. Зовнішні кордони світу обмежені стіною, яку не можна перетинати.

Сенсори зорового сприйняття робота не надто потужні, так само як і його інтелект. Зі свого поточного місцезнаходження він може бачити лише вміст сусідніх клітинок (на півночі, півдні, заході і сході) та поточної клітинки, яку він займає. Клітинка може бути порожньою, містити ресурс, або бути частиною стіни.

Протягом кожного сеансу роботи Роббі може виконати лише 200 дій. Кожна дія полягає у виборі однієї з можливих альтернатив: рухатись на північ, південь, захід або схід; рухатись у довільному напрямку; лишатись на місці; нахилятись і підіймати контейнер. За свої вчинки робот може отримати нагороду або бути покараним. Якщо він знаходиться на тій же клітинці,

в якій міститься контейнер, і підбирає його, то він отримує нагороду в 10 очок. Натомість, якщо він спробує забрати контейнер з порожньої клітинки, то втратить одне очко. Коли Роббі випадково робить неправильний крок і вдаряється об стіну, він отримує штраф у 5 очок і повертається на попередню клітинку. Найбільше очок протягом сеансу робот може заробити, якщо збере якнайбільше контейнерів, не вдаряючись у стіни та не нахиляючись дарма.

Т а б л и ц я . Фрагмент стратегії поведінки робота

Оточення					**Дія**
Північ	Південь	Захід	Схід	Поточна клітинка	
Порожньо	Порожньо	Порожньо	Порожньо	Порожньо	На Північ
Порожньо	Порожньо	Порожньо	Порожньо	Контейнер	Підняти Контейнер
Порожньо	Порожньо	Порожньо	Порожньо	Стіна	На Південь
…	…	…	…	…	…
…	…	…	…	…	…
Стіна	Стіна	Стіна	Стіна	Стіна	Не Рухатись

Потрібно представити стратегію поведінки робота, згідно якої він досягатиме певних результатів протягом сеансу роботи. У [17] запропоновано такий підхід: промоделювати усі можливі варіанти вмісту сусідніх клітинок та поточного положення робота й обрати для них деяку дію. Це і буде стратегією поведінки згідно з якою відбуватиметься рух світом протягом сеансу роботи робота. Фрагмент такої стратегії представлено в таблиці.

Для пошуку найкращої стратегії поведінки робота використаємо генетичний алгоритм. У якості особин популяції обрано множину стратегій. На початковому кроці стратегії поведінки генеруються випадковим чином. Генотипове представлення стратегії являє собою вектор з чисел від 0 до 6, кожне з яких відповідає одній з можливих дій відносно поточного стану навколишнього середовища. Оцінка пристосованості стратегії вираховується як середнє значення кількості очок, які заробить робот, використовуючи цю стратегію протягом 200 сеансів роботи. Сеанс роботи полягає у зборі випадковим чином розташованих контейнерів, початкове положення — точка (0,0). На кожен з 200 сеансів надається нова конфігурація розташування контейнерів, відмінна від попередніх. Розмір популяції стратегій було обрано рівним 100, що дозволило у підсумку отримати добре «тренованого» робота, який показував прийнятні результати після 1000 поколінь. Вибір особин для схрещування здійснювався пропорційно до значення пристосованості: чим вища пристосованість стратегії, тим більшою є ймовірність того, що вона буде обрана в якості одного з батьків. Було використано адаптацію одноточкового кросинговеру, який обмінював частини векторів дій між батьками до і після точки схрещування. Мутація особин полягала в тому, що з певною імовірністю випадковим чином обрані компоненти вектора змінювали своє значення.

«Актори–компоненти» алгоритму відпрацьовували у кластері з кількох екземплярів віртуальних машин JVM. Такий кластер може бути розгорнуто як і на одній потужній багатоядерній локальній машині, так і на кількох машинах, за умови наявності мережевого зв’язку між ними. Варто зауважити,

що фреймворк Akka самостійно контролює життєвий цикл кластера та окремих його вузлів. Від розробника вимагається лише запустити потрібні системи акторів на вузлах і коректно сформувати конфігурацію кластера. Конфігурація кластера, яку було використано у цій роботі, має такий вигляд:

```
akka {
  # loglevel = "OFF"
  actor {
    provider = "akka.cluster.ClusterActorRefProvider"
  }
  remote {
    netty.tcp {
      hostname = "127.0.0.1"
      port = 2051
    }
  }
  cluster {
    seed-nodes = [
              "akka.tcp://cluster@127.0.0.1:2051",
              "akka.tcp://cluster@127.0.0.1:2052",
               "akka.tcp://cluster@127.0.0.1:2053"
                 ]
    auto-down = on
  }
}
```

В масиві seed-nodes зберігаються хост-адреси та порти, на яких мають працювати вузли кластера. Також в конфігурації задається провайдер, котрий забезпечуватиме комунікацію між вузлами, в даному випадку це ClusterActorRefProvider. Докладний опис інших типи провайдерів та особливостей їхнього використання наведено у [18].

Визначальну роль в застосунку відіграє «актор-господар» (PopulationActor). Він відслідковує поточний стан популяції та вибирає особини, котрих надсилає допоміжним «акторам–робітниками». Екземпляри класів MutationActor та CrossoverActor виконують відповідні генетичні оператори над особинами, яких до них відправляє PopulationActor. Усі основні актори у застосунку імплементують інтерфейс Actor з фреймворку Akka та перевизначають метод receive, в якому описуються реакції на всі типи повідомлень, що можуть надходити даному актору. Також широко використовується взірець «допоміжний об’єкт», в якому описуються предметно-орієнтовані типи повідомлень, з якими працює той чи інший актор.

Для актора, котрий здійснює мутацію, реалізація «допоміжного об’єкту» виглядає таким чином:

```
object MutationActor {
  case class Mutate(robot: Robot)
  case class Mutated(robots: List[Robot])
}
```

Частковий клас Mutate містить робота-стратегію, окремі компоненти якої зазнають впливу мутації. У класі Mutated міститься список нових роботів-стратегій, отриманих у результаті мутації. За таким самим принципом створено об’єкт CrossoverActor, складові якого Cross та Crossed вміщують у собі батьків і повертають список породжених нащадків.

```
object CrossoverActor {
case class Cross(robot1: Robot, robot2: Robot)
case class Crossed(robots: List[Robot])
}
```

Наведемо імплементацію методу receive для PopulationActor, в якому реалізована головна логіка алгоритму:

```
def receive = {
case Broadcast =>
      broadcast ! population.head
      println(s"Best robot: ${population.head.points}, mutations:
              $mutations, crossovers: $crossovers")

case Mutated(robots) =>
      addToPopulation(robots)
      mutations += robots.length
      sender ! Mutate(randomRobot)

case Crossed(robots) =>
      addToPopulation(robots)
      crossovers += robots.length
      sender ! Cross(randomRobot, randomRobot)
}
```

Допоміжний об’єкт Broadcast використовується для друку в консоль пристосованості найкращої особини з поточного стану популяції. Отримавши повідомлення типу Mutated(robots), він обробляє особин, отриманих у результаті мутації та надсилає MutationActor нових особин. Схожим чином працює обробка повідомлення Crossed(robots), в якому містяться нащадки, породжені в результаті кросинговеру.

ВИСНОВКИ

У цій роботі здійснено огляд основних положень моделі акторів як підходу, що знову став актуальним відповідно до сучасних тенденцій розвитку апаратного забезпечення та вимог масштабування. У світлі появи нових гетерогенних архітектур апаратного забезпечення проаналізовано основні концепції побудови паралельних розподілених генетичних алгоритмів: «господар–робітники», грубозернисті, дрібнозернисті та гібридні моделі. Для концепції «господар–робітники», яка природнім чином дозволяє продемонструвати переваги моделі акторів, запропоновано адаптацію її синхронного та асинхронного варіантів до моделі акторів.

Використовуючи ідеї, втілені у Amazon Dynamo та розподілений базі даних Riak, засобами фреймворку Akka створено розподілену систему — кластер акторів. В середовищі кластера успішно розгорнуто застосунок, який демонструє використання пропонованої адаптації концепції «господар–робітники» для розв’язання задачі пошуку найкращої стратегії поведінки робота у штучному середовищі засобами моделі акторів.

Завдяки побудові застосунку продемонстровано гнучкість і технологічність моделі акторів у розподілених обчисленнях. Експеримент показав, що для великої кількості задач із невеликими об’ємами даних (дрібнозернисті задачі обслуговування великої кількості запитів у веб-сервісах) модель акторів зручніше та ефективніше, ніж важкі гріди.

Важливим напрямом подальших досліджень може стати поєднання моделі акторів з концепцією програмної транзакційної пам'яті (Software Transactional Memory). Таке поєднання може забезпечити альтернативний підхід до синхронізації обчислень, що дозволить розв'язати за допомогою моделі акторів широке коло задач.

ЛІТЕРАТУРА

1. *Hewitt C., Bishop P., Streiger R.* A Universal Modular Actor Formalism for Artificial Intelligence // IJCAI'73 Proceedings of the 3rd International Joint Conference on Artificial Intelligence. — Morgan Kaufmann Publishers Inc., San Francisco, 1973. — P. 235–245.
2. *Agha G.A.* ACTORS: A Model of Concurrent Computation in Distributed Systems. — MIT Press, Cambridge, Massachusetts, 1986. — 190 p.
3. *Gupta M.K.* Akka Essentials. — Birmingham: Packt Publishing, 2012. — 334 p.
4. *Wyatt D.* Akka Concurrency. — Walnut Creek, California, Artima Inc., 2013. — 515 p.
5. *Zheng L., Lu Y., Ding M.* Architecture-based Performance Evaluation of Genetic Algorithms on Multi/Many-core Systems // Proceedings of IEEE 14th International Conference on Computational Science and Engineering, Dalian, Liaoning, 24–26 Aug. 2011. — P. 321–334.
6. *Глибовец Н.Н., Медвидь С.А.* Генетические алгоритмы и их использование для решения задачи составления расписания // Кибернетика и системный анализ. — 2003. — № 1. — С. 95–108.
7. *Глибовець М.М., Гороховський С.С., Краткова О.В.* Гібридний генетичний алгоритм вирішення задачі оптимізації структури інтегральної схеми // Інженерія програмного забезпечення / Нац. авіац. ун-т — К.: НАУ. — 2011. — № 1. — С. 70–76.
8. *Глибовець М.М., Гулаєва Н.М.* Еволюційні алгоритми: підручник. — К.: НаУКМА, 2013. — 828 с.
9. *Haupt R.L., Haupt S.E.* Practical genetic algorithms. — Wiley-Interscience, 2004. — 272 p.
10. *Luque G., Alba E.* Parallel Genetic Algorithms: Theory and Real World Applications // Studies in Computational Intelligence. — Springer-Verlag, 2011, **367** — 172 p.
11. *Umbarkar A.J., Joshi M.S.* Review of Parallel Genetic Algorithm Based on Computing Paradigm and Diversity in Search Space // ICTACT Journal on Soft Computing. — 2013. — **3**, № 4. — P. 615–622.
12. *Nowostawski M., Poli R.* Parallel Genetic Algorithm Taxonomy // Third International Conference on Knowledge-Based Intelligent Information Engineering Systems, 1999. Proceedings. — P. 88–92.
13. *Бидюк П.И., Литвиненко В.И., Токарь А.А.* Параллельные генетические алгоритмы // Системні дослідження та інформаційні технології. — 2002. — № 4. — С. 7–16.
14. *Karmani R.K., Shali A., Agha G.* Actor frameworks for the JVM platform: a comparative analysis // PPPJ '09: proceedings of the 7th international conference on principles and practice of programming in java, Calgary, Alberta. — ACM, NY, 2009. — P. 11–20.
15. *Gupta* M.K. Akka Essentials. — Packt Publishing, Birmingham, 2012. — 334 p.
16. *Wyatt D.* Akka Concurrency. — Artima Inc., Walnut Creek, California, 2013. — 515 p.
17. *Mitchell M.* Complexity: A Guided Tour . — Oxford University Press, 2009. — 326 p.
18. *Akka*. Cluster Specification. Version 2.2.3. — http://doc.akka.io/docs/akka/2.2.3/common/cluster.html.

Надійшла 02.09.2014