# НОВІ ЗАСОБИ КІБЕРНЕТИКИ, ІНФОРМАТИКИ, ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ ТА СИСТЕМНОГО АНАЛІЗУ

УДК 681.3.06

**А.В. АНІСІМОВ**
Київський національний університет імені Тараса Шевченка, Київ, Україна,
e-mail: *avatatan@gmail.com.*

**О.В. ДЕРЕВЯНЧЕНКО**
Київський національний університет імені Тараса Шевченка, Київ, Україна,
e-mail: *olexandrder@gmail.com*.

**П.П. КУЛЯБКО**
Київський національний університет імені Тараса Шевченка, Київ, Україна,
e-mail: *kpp1@ukr.net*.

**О.М. ФЕДОРУС**
Київський національний університет імені Тараса Шевченка, Київ, Україна,
e-mail: *Alex.fedorus@gmail.com*.

## ТЕХНОЛОГІЯ PARCS: КОНЦЕПЦІЯ, РЕАЛІЗАЦІЇ, ВПРОВАДЖЕННЯ

**Анотація.** Наведено огляд розроблень за технологією PARCS (Parallel Asynchronous Recursive Control Space). Розглянуто концепцію керувального простору — модельного апарату, на основі якого описується логічна структура досліджуваної задачі (системи) і відображаються динамічні зміни в ній. Запропоновано PARCS-модель, застосування якої надає можливість гнучкої та уніфікованої адаптації до технологій програмування. Розглянуто PARCS-розширення мов програмування: PASCAL, C, FORTRAN, MODULA2, Java, CUDA, OpenCL, PYTHON, .NET, GO/PYTHON.

**Ключові слова:** керувальний простір, VPS, PARCS, розподілені системи, паралельне програмування, хмарні обчислення.

### ВСТУП

Основою парадигми послідовного програмування є значна множина абстрактних алгоритмічних моделей, наприклад машина Тюрінга, апарат частково-рекурсивних функцій тощо. Для паралельного програмування придатність алгоритмічних моделей значною мірою залежить від того, на якій платформі будуть функціонувати алгоритми. Наприклад, один і той самий алгоритм може мати абсолютно різну ефективність під час роботи на системах типу SIMD, MISD чи MIMD (Flynn's taxonomy).

Паралельна асинхронна рекурсивна керувальна система (PARCS-модель) є, образно кажучи, спільним знаменником для групи PARCS-технологій та PARCS-систем. Розроблення PARCS-моделі здійснюється на двох взаємно пов'язаних рівнях: логічному та програмному. На логічному рівні PARCS-моделі описується розподіл ресурсів та всі можливі схеми комутації та перекомутації зв'язків (з урахуванням рекурсивного розгортання алгоритму). Це рівень керувального простору (КП або CS), структура якого може з часом динамічно змінюватись. На програмному рівні описується керування, дані та правила взаємодії між даними та керуванням з урахуванням відповідної логічної структури, тобто здійснюється «наповнення» логічної структури. Такі дії є узагальненням ідей

дискретного перетворювача та рекурсивного перетворювача інформації, які були запропоновані В.М. Глушковим та його учнями і послідовниками в [1–11]. Особливо слід виокремити працю [5], в який вперше було описано PARCS-технологію.

## 1. КОНЦЕПЦІЯ КЕРУВАЛЬНОГО ПРОСТОРУ

Керувальний простір — це апарат, на основі якого описуються логічні структури досліджуваної задачі (системи) і відображаються динамічні зміни в ній. Структура КП складається з точок і каналів, які з'єднують точки, та має ієрархічну будову (у кожній точці на наступному рівні ієрархічного розгортання може бути новий КП (рекурсія за структурою КП). Точки і канали моделюють логічні процесори (ресурси) та логічні канали обміну інформацією. Побудову та модифікацію КП здійснюють (явно чи неявно) алгоритмічні модулі (АМ), які містяться в точках КП.

Розглянемо, як задається PARCS-модель:

- виокремлюється деякий набір базових алгоритмічних перетворювачів інформації (точніше, рекурсивних перетворювачів), тобто АМ;
- формується початкова структура КП, здійснюється його «наповнення» АМ;
- у процесі функціонування АМ в моделі допускається зміна КП (можливо, рекурсивна), створення нових активних копій АМ, «мутаційні» зміни нових копій АМ, які спричинені ситуаційною обставою, тобто загалом структура КП є динамічною;
- під час виконання АМ відбувається їхня взаємодія по каналах КП, причому передаватися чи прийматися можуть як дані, так і керування (використання та узагальнення ідей дискретного і рекурсивного перетворювачів інформації).

На основі PARCS-моделі — базової мови програмування та середовища реалізації, формується відповідна PARCS-технологія. На цьому етапі уточнюються такі аспекти: як передається керування, як буде реалізовано КП (точки та канали) — тільки засобами базової мови чи із застосуванням більш низькорівневих засобів тощо.

Значно вплинули на розроблення PARCS-концепції роботи класиків паралельних обчислень [12–14]. По суті PARCS-концепція є узагальненням, отриманих ними результатів. У теоретичному плані PARCS-концепція базується на теорії дискретних та рекурсивних перетворювачів інформації: кожен АМ є парою рекурсивних перетворювачів, які взаємодіють: один з них відіграє роль пам'яті, а інший — керування. Крім того, КП теж описується як рекурсивний перетворювач: структура КП — це керування, а наповнення точок КП тими чи іншими АМ — пам'ять [8].

Останній етап — це власне реалізація PARCS-системи. На цьому етапі важливими є фіксація базової мови, а також доповнювальних засобів, які можуть бути реазовані як на основі можливостей базової мови, так і на доповнені більш низькорівневими засобами.

## 2. ПЕРШІ РЕАЛІЗАЦІЇ PARCS-СИСТЕМ

Уперше PARCS-системи, а саме PARCS-PASCAL, PARCS-C, PARCS-FORTRAN [9–11], були реалізовані в середовищі мультикористувацької ОС RSX-11M (на комп'ютері СМ-4). Тут здійснювалось моделювання мультипроцесорної роботи у вигляді окремих задач для кожного процесу (окрема задача обслуговувала роботу КП). Першою було реалізовано PARCS-PASCAL [9], де паралелізм підтримувався за рахунок базової операційної системи. Мову PASCAL було розширено низкою операторів формування КП, приписуванням АМ у точку КП та передачею і прийомом повідомлень. Для рекурсивного розгортання нового

підпорядкованого КП викликалась відповідна процедура. Передача повідомлень виконувалась через спеціальний міжзадачний буфер. Подальші реалізації PARCS-C та PARCS-FORTRAN переважно повторювали початкову схему реалізації, але замість розширення базової мови новими операторами використовувались спеціальні процедури, що пришвидшувало час розроблення. Пізніше вже на основі MS-DOS (ПК) було реалізовано PARKS-MODULA2, тут розпаралелювання здійснювалось засобами мови MODULA2.

Реально мультипроцесорну конфігурацію було використано для реалізації PARCS-C на ПК з трансп'ютерною платою [15]. Планувальник розподіляв виконання АМ по трансп'ютерах. Фізична комутація трансп'ютерів між собою налаштовувалась перед початком роботи. Зрозуміло, що найбільшої ефективності обчислень вдавалося досягти, коли структура комутації трансп'ютерів була близькою до структури КП.

### 3. ВІРТУАЛЬНИЙ ПАРАЛЕЛЬНИЙ ПРОСТІР

Подальше розвинення технологій програмування і їхнє стрімке застосування у найрізноманітніших предметних галузях спричинило зростання уваги до систем з потужними обчислювальними можливостями. З'явилося чимало практично важливих задач з високими вимогами до часу та пам'яті. Одним із розв'язань цієї проблеми є ідея віртуального паралельного простору (virtual parallel space (VPS)) [16]. До складу VPS можуть входити як потужні мультипроцесорні комплекси, так і малопотужні термінали. Ці VPS можна розглядати як спеціальний випадок технології cloud computing, яка описана засобами PARCS-технології.

Реалізуються VPS на основі локальної чи глобальної комп'ютерної мережі, при цьому користувачі малопотужних терміналів мають змогу використовувати ресурси мультипроцесорних комплексів. Крім того, деякі задачі можуть бути розбиті на підзадачі для обчислення на різних вузлах комп'ютерної мережі. В обох випадках програмне забезпечення VPS підтримує процеси взаємодії різних вузлів мережі обміном повідомлень по мережі.

### 4. РОЗРОБЛЕННЯ PARCS-СИСТЕМ НА ОСНОВІ КОМП'ЮТЕРНИХ МЕРЕЖ

**4.1. Система PARCS-Java** [17–22]. Ця система реалізується на основі локальної чи глобальної комп'ютерної мережі і фактично є окремим випадком VPS. Застосування мови програмування Java надає можливість переносити програми між різними апаратними платформами, на яких встановлено Java віртуальну машину (JavaVM).

Позитивними якостями мови програмування Java [17, 18], які вплинули на вибір її як базової для проєктування системи паралельних обчислень, є такі:

- простота об'єктної моделі, завдяки чому легко проводити зміни в коді програми, доповнювати її та документувати;
- детермінованість коду — властивість мови, що запобігає появі помилок програміста, наприклад через такі несанкціоновані дії, як різні виклики;
- відповідність системи програмування Java сучасним вимогам, а саме підтримка засобів мережевого програмування (сокетів), потоковий ввід/вивід, графічний інтерфейс користувача, системи забезпечення пересилання даних по комп'ютерній мережі, технології, що дають змогу уніфікувати клієнт-серверний інтерфейс (RMI) тощо.

Недоліком мови Java є недостатня швидкість роботи JavaVM.

У середовищі PARCS-Java паралельні програми реалізуються на мережі, що складається з JavaVM. Це дає змогу використовувати паралельні програми

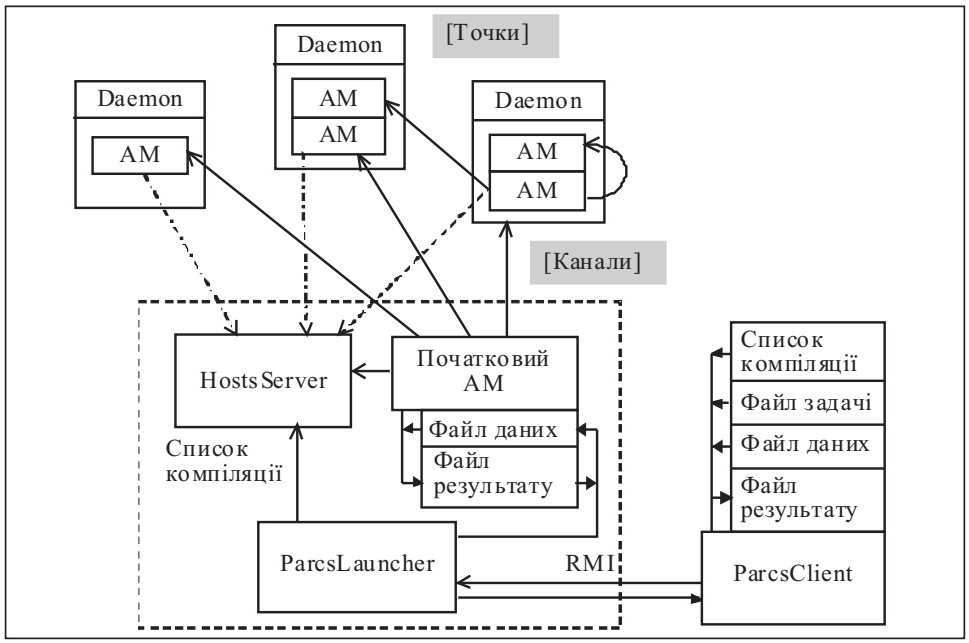


*Рис. 1.* Архітектура системи PARCS-Java

на довільних паралельних системах як однорідних, так і неоднорідних, а також на системах, що можуть складатися з вільних ресурсів Інтернету, для яких єдиною умовою є наявність JavaVM.

Архітектура системи PARCS-Java (рис. 1) складається з таких частин [17–20]:

- Parcs — основна бібліотека класів системи, яка містить всі основні класи, що використовуються для моделювання паралельних обчислень;
- Daemon (клієнт) — програма, яка встановлена на комп'ютері в мережі чи у хмарі і за допомогою якої проводяться обчислення;
- HostServer — сервер, в якому містяться список доступних клієнтів, а також інформація про поточні задачі та точки. Під час запуску модуль починає роздавати завдання клієнтам у залежності від кількості ядер процесорів (потужності) та їхньої завантаженості;
- AM — окремий проєкт (у вигляді алгоритмічного модуля), який описує алгоритм задачі, що обчислюється, та використовує бібліотеку Parcs.

На всіх машинах, що призначені для обчислень, запускається програма Daemon, а початковий AM та HostServer можуть запускатися або на тих самих, або на інших машинах. В окремому випадку навіть всі три елементи системи можуть бути запущені в одному місці.

У створеній реалізації зв'язок між клієнтом і сервером відбувається у такій послідовності:

- під час запуску модуля сервер перевіряє всі комп'ютери, зазначені у файлі hosts.txt;
- сервер розподіляє точки, що створюються в алгоритмічному модулі, по комп'ютерах, на яких виконується програма Daemon;
- на комп'ютері, де створена нова точка, початковий AM передає jar-файл (для PARCS-Java), в якому міститься вся інформація для виконання поточної задачі, і запускається процес обчислення;
- після завершення обчислень на всіх машинах Daemon відсилає результат назад до початкового AM, який обробляє всі результати і виводить загальний результат.

Для реалізації системи PARCS-Java у Cloud необхідно було додати підсистему автоматичного розгортання [20, 22]. Для цього було запропоновано Docker-технологію [21, 23] — технологію контейнеризації, яка дає змогу розробляти, розгортати і запускати програми в так званих контейнерах, а також є одночасно й інструментом розробника, і середовищем виконання. Ключовим компонентом Docker є реєстри — сховища образів. Реєстр дає змогу завантажити образи. Реєстри можуть бути публічними (наприклад, Docker Hub) і приватними.

У проєкті PARCS було створено і викладено в реєстр Docker Hub такі образи:

- Parcs.HostServer (*../parcshostserver*) для програми HostServer;
- Parcs.Daemon (*../parcsdaemon*) для програми Daemon;
- Parcs.Web (*../parcsweb*) для вебзастосунку для моніторингу роботи системи та ін.

Для більшості з сучасних реалізацій PARCS було створено відповідні образи для розгортання систем PARCS-Java [19, 24], PARCS-Python [24], PARCS.NET [24] та PARCS-Go/PARCS-Python [24] для локальної мережі та у хмарах.

**4.2. Система PARCS-CUDA.** Власне CUDA (Compute Unified Device Architecture) [25–27] — технологія GPGPU, що дає змогу програмістам реалізовувати алгоритми, які виконуватимуться на графічних процесорах компанії NVIDIA. Завдяки багатоядерній обчислювальній потужності графічних процесорів розробники можуть написати програмне забезпечення для розв'язання складних обчислювальних завдань за менший час.

Система PARCS-CUDA є розширенням технології PARCS [26], яке дає змогу писати CUDA-специфічні АМ для виконання на PARCS. Основні зміни стосуються Daemon (клас Daemon). Удосконалену архітектуру цього компонента наведено на рис. 2.

У класі Daemon (див. рис. 2) з'явився новий компонент Cudaemon. Якщо Daemon розпізнає АМ CUDA, то він передається для виконання Cudaemon, який працює в окремому потоці і в певний момент часу може виконувати лише один алгоритмічний CUDA-модуль або чекати нових завдань. Це зумовлено тим, що оптимальніше виконувати на одному CUDA-пристрої один хост-потік (інакше витрачається зайвий час на ініціалізацію CUDA-драйвера, а також на створення та синхронізацію потоків хосту). Звісно, що в такому контексті декілька АМ на одному хості будуть виконуватися вже не асинхронно, що потребує також деякої модифікації початкового АМ.

У реалізації системи PARCS-CUDA використано бібліотеку Java-біндингів для CUDA, а саме JCuda. Як реалізацію блокувальної черги завдань використано стандартний клас JDK: LinkedBlockingQueue. Для позначення, що АМ буде виконуватися на CUDA, введено анотацію на клас @CUDA.

Загальний алгоритм виконання Daemon на хості тепер такий:

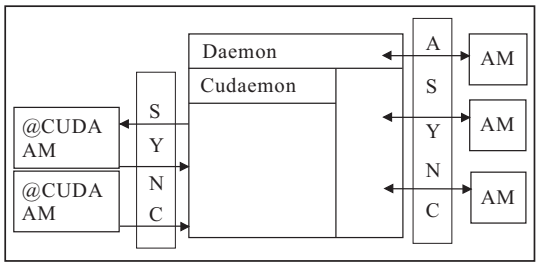


*Рис. 2.* Модифікована архітектура Daemon системи PARCS-CUDA

1) запуск, очікування повідомлення від HostServer, старт Cudaemon;

2) отримання (якщо потрібно) jar-файлу з АМ;

3) повідомлення про виконання АМ на відповідній точці (запускається новий потік; створюється нова точка і канал; якщо АМ не позначений @CUDA, то

він запускається в цьому ж потоці, інакше додається до черги Cudaemon; потік відразу завершується).

Час життя Cudaemon такий самий, як і звичайного Daemon — у ньому спочатку наявні команди ініціалізації CUDA, а після завершення — команди звільнення CUDA-ресурсів.

Така реалізація дає змогу запускати обчислювані задачі на GPU від компанії NVIDIA із застосуванням додаткових бібліотек, наприклад бібліотеки cuBLAS для задач обчислення множення матриць із застосуванням GPU.

**4.3. Система PARCS-OpenCL.** Відкрита мова обчислень OpenCL (Open Computing Language) — фреймворк для написання програм, що базуються на паралельних обчисленнях на різноманітних графічних та центральних процесорах [28, 29]. Фреймворк OpenCL складається з мови програмування, що базується на стандарті C99, та інтерфейсу програмування застосунків (API). Фреймворк OpenCL забезпечує паралелізм на рівні інструкцій та на рівні даних і фактично є реалізацією технології GPGPU та повністю відкритим стандартом. Його використання не обмежується ліцензійними угодами.

Розглянемо архітектуру підсистеми диспетчеризації задач та методи диспетчеризації в ній.

Система PARCS взаємодіє із підсистемою диспетчеризації задач (далі підсистемою) через інтерфейси IGlobalStrategy та ILocalStrategy.

Інтерфейс IGlobalStrategy реалізує глобальну стратегію розподілення задач між множиною доступних системі ресурсів. Вони виконують такі основні функції: аналіз нових завдань, підтримка роботи вже запущених завдань, збір інформації про ресурси (продуктивність, обсяг пам'яті, швидкість зв'язку тощо), балансування завантаження ресурсів. Це дає змогу повністю керувати стратегією диспетчеризації задач, не втручаючись безпосередньо в саму систему PARCS. Для зміни поведінки достатньо передати потрібну реалізацію інтерфейсу.

Інтерфейс ILocalStrategy реалізує локальні політики конкретних завдань. У різних задачах можуть бути потреби в різних ресурсах (CPU, пам'ять, швидкий мережевий зв'язок тощо). Задачі здатні передбачати хід свого виконання з різною деталізацією і точністю, мати якісь інші індивідуальні характеристики. Тому для різних завдань деякі стратегії працюють більш оптимально ніж інші. За рахунок цього, вказавши бажану локальну стратегію для завдання, можна отримати кращі результати.

Загальну архітектуру PARCS-системи наведено на рис. 3.

Опишемо процес виконання завдань. Нове завдання надходить із Інтернету до сервера PARCS (модуль HostServer). Завдання складається з множини задач, які необхідно розподілити між вузлами (комп'ютерами) та обчислити. Також завдання може містити (необов'язково) локальну стратегію планування (інтерфейс ILocalStrategy), яка найкраще задовольняє вимоги завдання. Сервер передає завдання глобальному диспетчеру задач, який аналізує завдання: ухвалює локальну стратегію завдання (залежно від глобальної політики він може прийняти запропоновану локальну стратегію, замінити на якусь іншу або взагалі керувати задачами завдання самостійно, тобто бути локальною стратегією для завдання), збирає дані, що надає завдання про процес його виконання, а далі залежно від вибраної стратегії і отриманої інформації здійснює планування, яке може бути обраховано як одразу, так і паралельно із виконанням самого завдання, розподіляючи задачі в реальному часі.

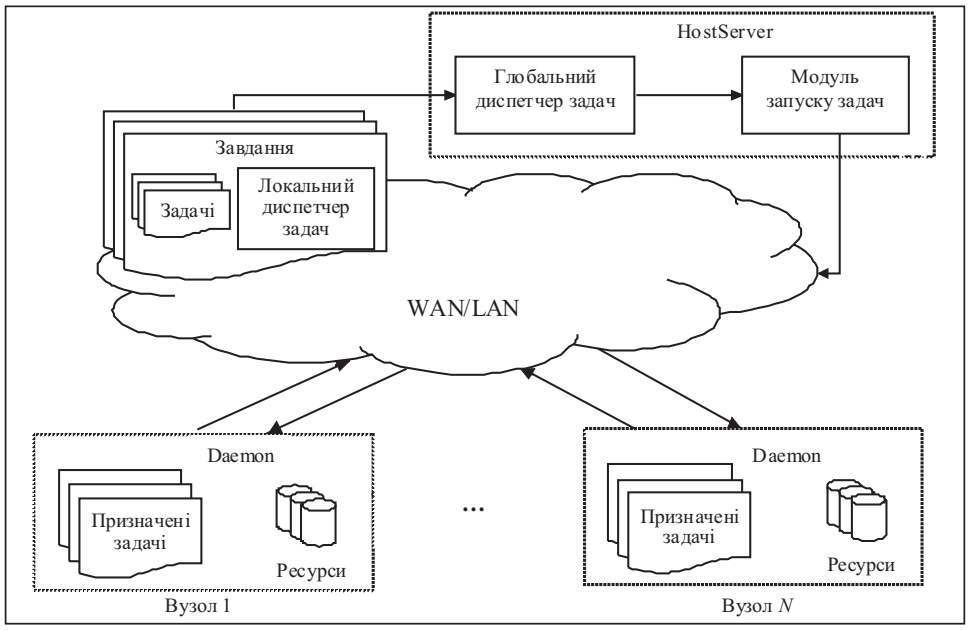


*Рис. 3*. Загальна архітектура PARCS-системи

Клас LSSimpleGPU є реалізацією локальної стратегії [22], яка розподіляє задачі за схожим принципом, який був жорстко прописаний в системі PARCS перед інтегруванням підсистеми диспетчеризації, але з деякими додатковими можливостями. Основна ідея полягала в динамічному балансуванні завантаження вузлів за таким критерієм: розподілення задач між вузлами пропорційно потужності вузлів. Тобто якщо один вузол вдвічі потужніший за інший, то він отримає вдвічі більше задач. При цьому не враховувалась складність задач, їхні потреби у пам'яті тощо. Додаткова модернізація, що була зроблена в LSSimpleGPU, порівняно з попереднім алгоритмом полягає у тому, що завдання за можливості має виконуватись на графічних процесорах. Тож, проводиться попереднє оброблення отриманої інформації про вузли, а саме про наявність у них можливості обчислювати завдання на графічних процесорах та інших обчислювачах. Це активно використовується для балансування навантаження на вузли. Якщо немає пристроїв з модулями графічних процесорів, то LSSimpleGPU працює так само, як попередній алгоритм.

У реалізації LSSimpleGPU продемонстровано, як можна запускати «системні» задачі на вузлах. Ця функціональність використовується для оцінювання наявності в них графічних процесорів, а також для вимірювання потужності вузлів. Системні задачі, зокрема, розв'язують великі системи лінійних рівнянь, вимірюючи продуктивність комп'ютерів, яка використовується для балансування навантаження.

Таким чином, реалізація LSSimpleGPU дає змогу запускати обчислювальні задачі як на CPU, так і на GPU в залежності від типу вибраної задачі та засобів (додаткових бібліотек) її розв'язання.

**4.4. Система PARCS-Python.** Опишемо реалізацію ідеї VPS [16] на прикладі PARCS-Python [30]. Така реалізація базується на локальній та глобальній комп'ютерній мережі, що дає змогу користувачам зі спільним hardware виконувати їхні програми на мультикомп'ютерних системах. Крім того, задача може бути розділена на підзадачі, кожна з яких буде виконуватись на окремій вершині, а завдання VPS буде полягати в організації взаємодії процесів між вершинами.

На рис. 4 наведено діаграму компонентів PARCS-Python, яка складається з точок КП, що поділяються на Master-вершину (МВ) та підлеглі вершини (Worker). За допомогою каналу МВ з'єднана з кожною підлеглою вершиною. РС має два протоколи: HTTP і RPC (ZeroRPC реалізація). Вершини використовують протокол HTTP для комунікації, відправляючи та отримуючи інструкції з метаінформацією. Протокол RPC дає змогу відправляти до АМ інструкції та отримувати результат виконання.

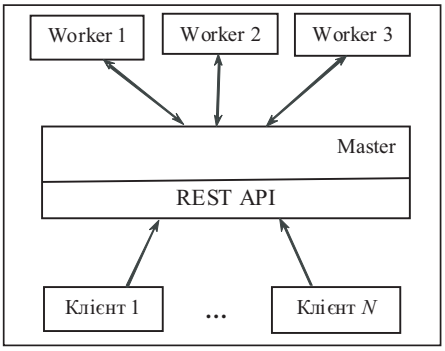


*Рис. 4.* Діаграма компонентів PARCS-Python

Згадана МВ підтримує REST API, яка допомагає користувачам взаємодіяти з PARCS-системою. Немає ніяких обмежень для вершини: вона може розташовуватись на одному комп'ютері чи на кількох комп'ютерах, з'єднаних в мережу. Єдине обмеження — не має бути конфліктів на дисковому рівні

Наведемо переваги такої архітектури:

- внутрішній стан системи є інкапсульованим (клієнтський код не має нічого знати про топологію системи, окрім наданої інформації);
- система є відмовостійкою, тобто МВ відповідає за оброблення помилок;
- Python — це мова, яка інтерпретується, отже немає ніяких обмежень, характерних для мов компіляції.

Однак така реалізація приховує складність системи, а клієнтський код не може модифікувати її топології.

Розглянемо життєвий цикл запиту.

1. МВ отримує запит і ставить його в чергу задач.
2. Процес демона-планувальника (Scheduler daemon), який виконується на МВ, вибирає запит із черги і починає його оброблення, причому першим кроком оброблення є розподілення клієнтського коду між вершинами.
3. Код МВ ініціює клієнтський код інформацією про системну топологію і обчислює отримані дані; зазвичай клієнтський код зчитує дані, виділяє фрагменти та виконує крок map алгоритму map-reduce.
4. Підпорядковані вершини отримують свої фрагменти даних та обчислюють їх і повертають результати МВ за протоколом RPC.
5. Клієнтський код МВ отримує результати від підлеглих вершин і виконує крок reduce.
6. Опціонально клієнтський код може отримати результат map-reduce цього кроку і виконати ще один map-reduce над новими даними.
7. Клієнтський код записує остаточний результат на диск.
8. Планувальник звільняє всі ресурси, які використовувались під час оброблення поточного запиту і вибирає новий запит із черги.

**Протоколи PARCS-Python та API.** Зазвичай компоненти PARCS-системи розміщені в мережі. Реалізація Python PARCS-системи використовує RPC та HTTP як базові протоколи для комунікації між вершинами та між клієнтами і вершинами, а саме:

- протокол RPC (його ZeroRPC-реалізація) використовується системою, а клієнт не може отримати безпосередній доступ до нього;
- протокол HTTP (REST API) має два типи REST API: public та private. Public REST API (табл. 1) може бути використаний клієнтами для запиту інформації про стан системи. МВ також використовує private REST API (табл. 2) для розподілення інформації та клієнтських файлів між вершинами.

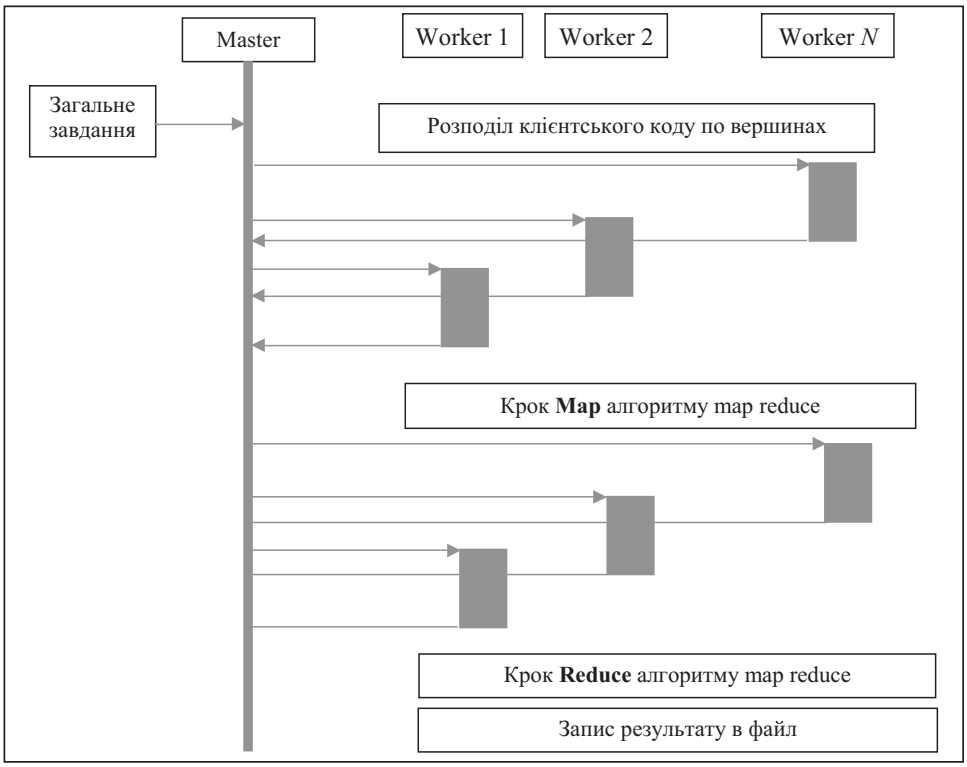


*Рис. 5.* Життєвий цикл запиту

**Таблиця 1.** Public REST API

| HTTP метод | HTTP URI | Тіло запиту | Опис |
|---|---|---|---|
| POST | /api/jobs | {"name":"Job Name","solution": @solution file, "input": @input file} | Post a job |
| GET | /api/jobs | | Get job list |
| GET | /api/job/: job id | | Get information about specific job |
| GET | /api/nodes | | Get slave node list |
| GET | /api/nodes/: node id | | Get information about specific slave node |

**Таблиця 2.** Private REST API

| HTTP метод | HTTP URI | Опис |
|---|---|---|
| POST | /api/internal/node | Connect to master node with slave node |
| POST | /api/internal/job | Master node distribute client code over slave nodes |

Протокол PARCS-Python із зручним вебінтерфейсом відкриває нові перспективи щодо розроблення та впровадження розподілених алгоритмів із застосуванням хмарних технологій.

**4.5. Система PARCS.NET.** Архітектура цієї системи [30–33] в основному складається з таких самих частин, що й архітектура системи PARCS-Java, тобто Parcs, Daemon, HostServer та AM.

Додатково використовується також Web — окремий вебзастосунок для моніторингу поточного стану виконання задач, який також дає змогу завантажувати і запускати свої АМ. Функціонування теж в основному схоже на функціонування системи PARCS-Java.

Було розроблено новий, кращий для хмарних обчислень алгоритм комунікації клієнта і сервера. Зараз HostServer не повинен знати про всі машини, на яких запущено Daemon. Натомість Daemon під час запуску з'єднується з HostServer і передає йому всю необхідну інформацію про себе. Таким чином, можна легко додавати нові обчислювачі до кластера, не вносячи змін у конфігурації HostServer.

**4.6. Система PARCS-WCF.** Натепер система PARCS-Java є найпопулярнішою і найпоширенішою реалізацією PARCS. Порівняно з PARCS-Java система PARCS-WCF [34, 35] має такі особливості:

- наявність асинхронних інтерфейсів;
- краще використання мережевих ресурсів. У PARCS-Java кожна точка прив'язана до деякого сокету та обмін даними відбувається під час установлення контакту. За такого підходу одна з точок постійно намагається підключитися до іншої або очікує інформацію. Система PARCS-WCF використовує два сокети: один для Daemon, інший для сервісу точок. Передача даних між точками відбувається через сервіс точок і дані завжди передаються. Таким чином, блокування відбувається тільки під час очікування даних;
- оптимізація передачі даних. За можливості передачі даних усередині машини система PARCS-WCF взагалі не виходить на рівень TCP-IP. Користувач може самостійно вибирати засоби серіалізації даних;
- додаткова можливість рекурсивно створювати точки.

Важливим сценарієм роботи системи PARCS-WCF є розподілення обчислень по різних машинах. Особливістю такого режиму роботи є те, що виключення, отримані в результаті роботи, в звичайному випадку не будуть показані IDE і у разі виникнення помилки у деякому рекурсивному модулі її буде важко знайти. Для налагодження розподілених застосунків та виправлення помилок у рекурсивних (динамічно) створюваних модулях можна скористатися цією особливістю PARCS. Кожну точку в будь-який момент часу можна описати трійкою: $\langle$*Internal state*, *Data*, *Channels*$\rangle$, де *Internal state* — внутрішній стан точки, отриманий в результаті роботи АМ, *Data* – набір даних, який був переданий у точку, та *Channels* — сукупність доступних точці каналів. При цьому *Internal state* для більшості алгоритмів буде змінюватись лише між отриманням нових даних з інших точок, тобто він повністю залежить від *Data*. Своєю чергою *Channels* можуть змінюватись в будь-який момент, проте на практиці це відбувається винятково у більшості алгоритмів. Наявність *Channels* має значення лише під час відправлення (отримання) повідомлень.

Отже, можна дійти висновку, що точку можна привести повторно у стан відмови, знаючи дані, що до неї надходили, і їхню послідовність. У налагодженні та пошуку помилок у розподілених (паралельних) застосунках найскладнішим є відтворення умов, за яких можна виявити помилку. У випадку використання PARCS достатньо знати АМ та логувати дані (канали), що надходять у точку, для повного відтворення стану, у якому виникла помилка. За звичайних умов отримати ці логи було б складно, оскільки кожна точка зберігала би їх окремо, але, оскільки система PARCS-WCF передає дані через власні сервіси, можна реалізувати логування всередині них. Звісно є деякі обмеження на алгоритми в активному АМ, наприклад, якщо алгоритм використовує

випадкові значення або існує якась залежність від часу роботи чи від деяких інших зовнішніх чинників, цей метод не спрацює.

**Реалізації системи PARCS-WCF мовою C#.** Ця реалізація може бути використана різними мовами платформи .Net. Вона послуговується технологією WCF, яка не є мультиплатформною, і поки немає можливості її перенесення на інші платформи (паралельно з удосконаленням .Net Core). Але, оскільки демони є WCF-сервісами, їх можна динамічно створювати в хмарному середовищі.

**4.7. Гетерогенна PARCS-система**. На практиці одною з проблем PARCS-систем була несумісність і непереносимість наявних АМ між мовами програмування. Наприклад, у разі намагання запустити два різні алгоритми, які написані на різних мовах на одному кластері комп'ютерів (мережа чи хмара), потрібно запускати дві копії програмного забезпечення для функціонування PARCS. Отже, сервер PARCS-Java вміє керувати лише демонами PARCS-Java, а PARCS.NET — відповідно тільки своїми демонами. Це також не дає змоги написати алгоритм, який використовуватимуть одночасно ці два алгоритми, оскільки не зрозуміло, на якій мові його писати і тим більше — запускати.

Далі пропонується підхід, що уможливлює об'єднання кількох імплементацій системи PARCS, як от Heterogeneous PARCS-Go/PARCS-Python об'єднати в одну уніфіковану версію, що буде складатися з сервера та демона, які можуть підтримувати більшість сучасних мов програмування за допомогою бібліотек.

**Особливості гетерогенної PARCS-системи.** Розглянемо проблеми (та можливості їхнього розв'язання), що виникають під час роботи з конкретними PARCS-системами:

- наявність окремих серверів (демонів) для кожної мови;
- неможливість співпраці між різними мовами;
- неізольоване середовище виконання;
- складність у використанні сторонніх залежностей;
- потреба у новій імплементації PARCS через додавання мови;
- відсутність уніфікованого графічного інтерфейсу;
- складність розгортання кластера.

Уявімо, що створюється якийсь код, який викликає сервіс, що має REST API або Thrift API. Для того щоб зробити запит, код повинен враховувати мережеве розташування (IP-адресу та порт) екземпляра сервісу. У традиційному застосунку, запущеному на фізичному обладнанні, мережеве розташування екземплярів сервісу є відносно статичними. Наприклад, розроблюваний код може читати мережеві розташування з конфігураційного файлу, який періодично оновлюється. Проте, у сучасних мікросервісах, що базуються на хмарних технологіях, розв'язати цю проблему набагато складніше [36].

Екземпляри сервісів мають динамічно призначені мережеві розташування. Мало того, набір екземплярів одного сервісу динамічно змінюється через автомасштабування, збої та оновлення. Отже, клієнтський код повинен використовувати більш складний механізм виявлення сервісу.

Існує два основних шаблони виявлення сервісів: з боку клієнта та з боку серверів [36].

Дизайн та розроблення нової гетерогенної PARCS-системи підтримує:

- кооперацію АМ, написаних різними мовами програмування;
- ізоляцію середовища виконання;
- зручний спосіб керування залежностями;
- можливість додавання будь-якої мови програмування;
- підтримку кількох мов програмування: Python та Go (натепер);
- багатофункціональний графічний вебінтерфейс;
- автоматичну підтримку будь-якого Docker Swarm-кластера.

Docker Swarm-технологія [37] дає змогу досягти можливості перенесення необхідного програмного забезпечення та легкість у налаштуванні кластера через такі фактори:

- постачається разом зі стандартним Docker Engine;
- простий в налаштуванні (порівняно з Kubernetes);
- переймає задачу розподілення ресурсів та планування;
- гарантує ізоляцію всередині кластера.

**Архітектура мовно незалежної PARCS-системи.** Побудова цієї архітектури ґрунтується на відомій ідеї PARCS-системи, але замість того, щоб запускати задачі в різних потоках операційної системи, запускаються АМ, написані різними мовами в окремих процесах ОС. Таким чином, позбуваються спільного знаменника — мови для реалізації алгоритму. Достатньо уніфікувати шлях передачі даних через канали так, щоб програми, написані різними мовами, могли «спілкуватися» між собою. Для цього вже існують готові рішення і формати передачі та представлення даних, наприклад JSON, MessagePack, Thrift, Protocol Buffers та навіть XML. Усі вони розв'язують одну задачу — серіалізацію даних у формат, що можна буде потім десеріалізувати. Оскільки типізація повідомлення не суттєва і вигідніше використовувати ефективні бінарні формати, пропонується в конкретній реалізації використати MessagePack.

Алгоритм роботи такий: коли АМ1 хоче виконати АМ2, то відбувається запит до сервера, який відповідає за розподілення роботи; сервер повертає адресу комп'ютера в мережі та порт, за яким буде працювати процес, що відповідає АМ2; через цей канал буде відбуватися подальша комунікація між модулями.

## ВИСНОВКИ

PARCS-технології та відповідні PARCS-системи дають можливість організувати паралельні обчислювальні процеси, базуючись на різних програмних платформах [22, 24, 30–33, 35]. Розглянуті у статті засоби PARCS надають користувачу такі можливості:

- підтримка складних обчислювальних задач, які потребують великих обчислень та паралельного оброблення даних;
- керування інформаційними потоками в системах паралельного оброблення;
- накопичення алгоритмів паралельного програмування (у вигляді АМ) для їхнього подальшого використання;
- застосування хмарних технологій [38–40] для розв'язання складних обчислювальних задач, а саме легкість у налаштуванні та використанні, можливість швидкого запуску застосунків і створення великої кількість віртуальних машин у хмарі для розподілених обчислень, налаштування безпеки та мережевого доступу.

У [24] розглядаються приклади різних реалізацій PARCS-систем у хмарах.

## СПИСОК ЛІТЕРАТУРИ


1. Глушков В.М., Летичевский А.А. Теория дискретных преобразователей. В кн. Избранные вопросы алгебры и логики. Новосибирск: Наука, СО, 1973. С. 5–39.
2. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Автоматизация проектирования вычислительных машин. Киев: Наук. думка, 1975. 232 с.
3. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Теория структур данных и синхронные параллельные вычисления. *Кибернетика*. 1976. № 6. С. 2–15.
4. Глушков В.М., Капитонова Ю.В., Летичевский А.А., Горлач С.П. Макроконвейерные вычисления функций над структурами данных. *Кибернетика*. 1981. № 4. С. 13–21.

5. Глушков, В.М., Анисимов, А.В. Управляющие пространства в асинхронных параллельных вычислениях. *Кибернетика*. 1980. № 5. С. 1–9.

6. Глушков В.М., Капитонова Ю.В., Летичевский А.А. Алгебра алгоритмов и динамические распараллеливание программ. *Кибернетика*. 1982. № 5. С. 4–10.

7. Гороховский С.С., Капитонова Ю.В., Летичевский А.А., Молчанов И.Н., Погребинский С.Б. Алгоритмический язык МАЯК. *Кибернетика*. 1984. № 3. С. 54–74.

8. Анисимов А.В. Рекурсивные преобразователи информации. Київ: Вища шк., 1987. 392 с.

9. Анисимов А.В., Кулябко П.П. Программирование параллельных процессов в управляющих пространствах. *Кибернетика*. 1984. № 3. С. 79–88.

10. Анисимов А.В., Борейша Ю.Е., Кулябко П.П. *Программирование*. 1991. № 6. С. 91–102.

11. Анисимов А.В., Кулябко П.П. Особенности ПАРУС-технологии. *Кибернетика и системный анализ.* 1993. № 3. С. 128–137.

12. Hoare C.A.R. Communicating sequential processes. *Communications of the ACM*. 1978. Vol. 21, N 8. P. 666–677.

13. Dijkstra E.W. Co-operating sequential processes. In: Programming Languages: NATO Advanced Study Institute: Lectures Given at a Three Weeks Summer School Held in Villard-le-Lans, 1966. Ed. by F. Genuys. Academic Press Inc., 1968. P. 43–112.

14. Brinch Hansen P. Distributed processes: a concurrent programming concept. *Communications of the ACM.* 1978. № 11 P. 666–677.

15 Гриценко Д.М. Методы и средства программирования для транспьютерных комплексов: дис. … канд. физ.-мат. наук. Ин-т кибернетики им. В.М. Глушкова НАН Украины. Киев, 1996. 121 с.

16. Анісімова Л.А., Кулябко П.П., Вєтров А.М., Шевченко В.П. Керуючий простір як віртуально паралельний простір. *Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки.* 1999. № 4. С. 82–86.

17. Анисимов А.В., Деревянченко А.В. Система ПАРУС-JAVA для параллельных вычислений на компьютерных сетях. *Кибернетика и системный анализ*. 2005. № 1. С. 25–36.

18. Дерев'янченко О.В. Моделювання паралельних програм за допомогою системи ПАРКС-JAVA. *Наукові записки НаУКМА. Комп'ютерні науки*. 2005. Т. 36. С. 47–58.

19. Анісімов А.В., Деревянченко О.В. Паралельне програмування із застосуванням технології ПАРКС-JAVA. Київ: ТОВ «Компанія ВАІТЕ», 2013. 78 с.

20. Деревянченко О.В. Системи паралельних обчислень на комп'ютерній мережі на базі технології ПАРКС. *Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки.* 2014. № 2. С. 124–127.

21. Docker — контейнерна платформа. URL: https://www.docker.com/.

22. Деревянченко О.В., Сакевич Р.Д. Застосування системи ПАРКС-Java та Google Cloud Platform для хмарних обчислень. *Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки.* 2017. № 4. С. 69–74.

23. GitHub — репозиторій PARCS-JAVA. URL: https://github.com/lionell/labs/tree/master/parcs.

24. Деревянченко О.В. Налаштування системи ПАРКС для хмарних обчислень: Методичні рекомендації для студентів факультету кібернетики. Київ, 2018. 60 с. URL: http://parcs.unicyb.kiev.ua/mr/PARCS_MR.pdf.

25. NVIDIA CUDA™ C Programming guide 4.2. URL: http://developer.download.nvidia.com/compute/ DevZone/docs/html/C/doc/CUDA_C_Programming_Guide.pdf.

26. Java bindings for CUDA. URL: http://www.jcuda.org.

27. Nickolls J., Buck I., Garland M., Skadron K. Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for? *Queue*. 2008. Vol. 6, N 2. P. 40–53. https://doi.org/10.1145/1365490.1365500.

28. OpenCL™ — open standard for parallel programming of heterogeneous system. URL: http://www.khronos.org/opencl/.
29. Java bindings for OpenCL. URL: http://www.jocl.org/.
30. Anisimov A.V., Kuliabko P.P., Hodovaniuk V.I. Parallel programming in computer networks on the base of PARCS-technology (basic language is Python). *Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки*. 2016. № 3. С. 57–60.
31. Деревянченко О.В., Хавро А.Ю. Застосування системи ПАРКС.NET та Amazon EC2 для хмарних обчислень. *Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки*. 2015. № 4. С. 111–118.
32. Анісімов А.В., Деревянченко О.В., Хавро А.Ю. Застосування системи ПАРКС.NET в Docker контейнерах та Google Cloud Platform для розподілених хмарних обчислень. *Штучний інтелект.* 2018. Т. 81, № 3. С. 52–61.
33. Anisimov A.V., Derevianchenko O.V., Kuliabko P.P., Fedorus O.M. Programming system PARCS. *Journal of Computer and Communications.* 2017. Vol. 5, N 9. P. 129–139.
34. Федорус О.М. Застосування ПАРКС-C# для моделювання паралельно-рекурсивних процесів. *Вісник Київського національного університету імені Тараса Шевченка. Серія фізико-математичні науки.* 2017. № 1. С. 75–79.
35. Анисимов А.В., Федорус А.М. Разработка и перспективы системы ПАРУС-WCF. *Кибернетика и системный анализ*. 2020. Т. 56, № 1. С. 178–185.
36. Service discovery in a microservices architecture. URL: https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/.
37. Docker Swarm — керування кластером Docker Engine. URL: https://docs.docker.com/engine/swarm/.
38. Google Cloud Platform. URL: https://cloud.google.com.
39. Amazon Elastic Compute Cloud (Amazon EC2). URL: https://aws.amazon.com/ec2/.
40. Microsoft Azure. URL: https://azure.microsoft.com/.

**A.V. Anisimov, O.V. Derevianchenko, P.P. Kuliabko, O.M. Fedorus**

**PARCS TECHNOLOGY: CONCEPT AND IMPLEMENTATIONS**

**Abstract.** An overview of PARCS (Parallel Asynchronous Recursive Control Space) technology developments is provided. The concept of the control space — a model apparatus, based on which the logical structure of the investigated problem (system) is described and dynamic changes in it are reflected, is considered. The PARCS model is proposed, whose application leads to flexible and unified adaptation to emerging programming technologies. PARCS-extension of programming languages is considered: PASCAL, C, FORTRAN, MODULA2, Java, CUDA, OpenCL, PYTHON, .NET, GO/PYTHON..

**Keywords:** CS, VPS, PARCS, distributed systems, parallel programming, programming languages, cloud computing.

*Надійшла до редакції 15.05.2023*