УДК 658.012.011.56:378](477)

Глибовець М.М., Гороховський С.С., Стукало М.С.

ПРОБЛЕМИ КООРДИНАЦІЇ В РОЗПОДІЛЕНИХ ОБЧИСЛЕННЯХ

В роботі проаналізовані дві основні моделі координації в розподілених обчисленнях: data-driven (стан обчислень на будь-який момент часу визначається даними, що отримуються або передаються та фактичною конфігурацію скоординованих компонентів) та control-driven (в основу координації покладено спільний простір даних, який є асоціативною структурою даних). Крім виділених стилістичних відмінностей між цими координаційними моделями, які впливають на ступінь розділення обчислювальних та координаційних частин, зазначено і різне застосування. Приведений короткий опис координаційної класичної моделі Linda та її імплементацій, показує простоту привнесення координаційних механізмів до будь-якої мови програмування.

***Ключові слова**: розподілені системи, data-driven, control-driven, координаційна модель Лінда.*

В работе проанализированы две основные модели координации в распределенных расчетах data-driven (состояние расчетов на любой момент времени определяется данными, которые получаются или передаются и фактической конфигурацией скоординированных компонентов) и та control-driven (в основу координации положено общее пространство данных, которое является ассоциативной структурой данных). Кроме выделенных стилистических отличий между этими координационными моделями, которые влияют на степень разделения расчетных и координациооных частей, обозначено и разное применения. Приведенное короткое описание координационной классической модели Linda и ее имплементаций, показывает простоту внесения коородинационных механизмов в любой язык программирования.

***Ключевые слова:** распределенные системы, data-driven, control-driven координационная модель Линда.*

Two major models of coordination in distributed systems are analyzed: data-driven (the state of computation at any moment of time is determined by the data received or transmitted and by the actual configuration of the coordinated components) and control-driven (a shared data space in the form of an associative data structure is the basis of the coordination). Besides the named stylistic differences between the coordination model, which impact the degree of distribution between computational and coordination parts, some differences in applications are also considered. The given brief description of the classic coordination model Linda and of its implementations demonstrates the simplicity of injecting coordination tools into any programming language.

***Key words:** distributed systems,data-driven, control-driven, coordination model Linda.*

ВСТУП

Використання розподілених паралельних систем – важливий напрям розвитку обчислювальних пристроїв спрямованих на ефективне вирішення актуальних задач сьогодення: моделювання процесів згортання/розгортання молекул білку, прогнозування змін клімату, вирішення проблеми керованого термоядерного синтезу, створення нових напівпровідникових та композитних матеріалів, робіт у галузі машинного інтелекту і робототехніки, фінансового моделювання та торгівлі, створення складної тривимірної анімації та комп’ютерних спецефектів [1; 2].

У більшості варіантів розпаралелення та забезпечення комунікації між процесорами, що беруть участь у обчисленнях, використовують спеціалізовані прикладі програмні інтерфейси

API (Application Programming Interface). Окремо, тут можна виділити інтерфейс передачі повідомлень MPI (Message Passing Interface). Назва «інтерфейс передачі повідомлень» говорить сама за себе. Це добре стандартизований механізм побудови паралельних програм у моделі обміну повідомленнями. Існують стандартні «прив'язки» MPI до мов C/C++, Fortran 77/90. Безкоштовні та комерційні реалізації є майже для всіх суперкомп'ютерних платформ, а також для мереж робочих станцій UNIX і Windows. У даний час MPI найбільш широко використовується і динамічно розвивається серед представників свого класу [3].

Використання всього потенціалу масштабних розподілених систем вимагає наявності програмних моделей, які явно оперують поняттями паралельної співпраці між дуже великим числом активних сутностей, які складають єдине застосування. Така потреба призвела до проектування та впровадження декількох координаційних моделей разом з окремими мовами, які підтримують ці моделі. Майже всі ці схеми створювалися для того, щоб надати розробникам каркас (framework), який би покращував модульність, сприяв би повторному використанню компонентів (послідовних чи вже паралельних), підвищував би платформну незалежність та мовну сумісність. Однак, ці моделі відрізняються в тому, як саме визначається поняття координації, що конкретно координується, якими засобами досягається координація, і які метафори застосовані для представлення цих понять.

Розробка та впровадження великих і складних систем, таких як авіадиспетчерська служба, транспортна навігація, інтелектуальний пошук та обробка даних, показали, що єдина мова програмування чи системна архітектура не в змозі охопити всі можливі ситуації, які виникають при розробці такого складного та багатофункціонального застосування. Підхід до програмування великих застосувань було знайдено в ідеях багатомовності та гетерогенності. Багатомовне або мультипарадигменне програмування підтримує декілька різних парадигм програмування, забезпечує зв'язок між ними та ізолює небажану взаємодію. Під гетерогенністю мається на увазі те, що мова програмування для гетерогенних систем повинна підтримувати багато різних моделей обчислень [4; 5].

Концепція координації тісно пов'язана з багатомовністю та гетерогенністю. Розробку паралельного чи розподіленого застосування можна представити як розробку двох окремих частин: власне обчислювальна частина об'єднує процеси, відповідальні за обробку даних, а координаційна частина бере на себе комунікацію та кооперацію між процесами. Тому координацію можна використати для відділення обчислень в розподіленому чи паралельному застосуванні від власне комунікації, що дозволяє проводити розробку двох частин окремо і потім об'єднати результати роботи. Оскільки координаційний компонент системи відділено від обчислювального, то останній можна вважати «чорною скринькою», і тому конкретна мова, використана для обчислень, не має значення для координаційного механізму. І оскільки мови для обчислень можуть бути різними, то координація заохочує використовувати гетерогенні архітектури.

Девід Гелернтер та Ніколас Кареро дали найбільш поширене та загальноприйняте визначення координації. *Координація – це процес створення програм за допомогою склеювання активних фрагментів* [6]. Відповідно, координаційна модель – це той клей, який склеює окремі процеси в єдиний ансамбль [5].

Координаційні моделі можна розділити по багатьом критеріям, але найбільш поширена класифікація на дві категорії: data-driven (орієнтовані на задачу) або control-driven (орієнтовані на процеси) координаційні підходи.

Аналізу цих моделей і присвячена дана робота.

1. ВИМОГИ ДО СИСТЕМ КООРДИНАЦІЇ

У класичній роботі [4] визначено базові вимоги до координаційних моделей. Серед них виділяється задоволення таких властивостей, як ортогональність, простота та загальність.

У програмуванні під *ортогональністю* розуміється поєднання незалежних концепцій. Наприклад, для паралельного програмування необхідна обчислювальна та координаційна

моделі, які є ортогональними або незалежними. Кожна модель відповідає за свою частину роботи, не втручаючись у сферу відповідальності іншої моделі. Такого типу ортогональність робить можливим те, що процес-споживач інформації не мусить мати певних наперед визначених знань про відправника інформації.

Наслідком комунікаційної ортогональності є можливість розділення часового та просторового аспектів обміну інформацією, тобто використання розподіленої спільної пам'яті. Просторове розділення означає те, що процеси можуть обмінюватися інформацією через спільний простір даних без жодної інформації про те, хто ввів кортеж у цей простір. Часове розділення дозволяє двом процесам спілкуватися через спільний простір даних без вимоги одночасної їх активності.

Вимога *простоти* стосується в основному комунікації між процесами. Моделі координації, орієнтовані на задачі (такі, як Linda), використовують спільний простір даних, що надає розробникам простий спосіб для досягнення різних цілей, головна серед яких – спілкування між процесами. Комунікація процесів між собою через спільний простір даних підтримується завдяки невеликій кількості операцій, які можуть бути легко використані програмістом.

У середині 1980-х років Девід Гелернтер з Йельського університету запропонував скоріше підхід, ніж окрему мову, для паралельного програмування, що називалася Linda [4]. Ідея побудови системи Linda виключно проста, а тому красива і дуже приваблива. Паралельна програма складається з багатьох паралельних процесів, кожен з яких працює як звичайна послідовна програма. Усі процеси мають доступ до спільної пам'яті, одиницею зберігання в якій є кортеж. Для зв'язку процесів і спільної пам'яті існує всього шість операцій і у більшості випадків цього цілком достатньо.

Загальність описує можливість і бажання визначати координаційну модель таким чином, щоб її можна було застосовувати у будь-якій множині асинхронних застосувань, таких як розподілені системи.

1.1. Модель data-driven координації

Основною особливістю моделей та мов data-driven координації є той факт, що стан обчислень на будь-який момент часу визначається даними, що отримуються або передаються та фактичною конфігурацію скоординованих компонентів. Іншими словами, координатор чи координований процес відповідає як за перевірку і маніпулювання даними, так і за координацію себе і/або інших процесів за допомогою координаційних механізмів, які забезпечує кожна мова. Це не виключає існування чіткого поділу координаційних механізмів та чисто обчислювальних функцій деякого процесу. Але, як правило, це значить, принаймні стилістично і лінгвістично, що існує суміш координації та обчислень у коді процесу.

Мови data-driven координації, як правило, надають деякі координаційні примітиви (у поєднанні з координаційною метафорою), які змішуються з чисто обчислювальною частиною коду. Ці примітиви зручно інкапсулюють комунікаційні і конфігураційні аспекти деяких обчислень, але вони повинні бути використані у поєднанні з чисто обчислювальною обробкою даних, пов'язаних з процесом. Це означає, що про процес не можна легко визначити, координаційний він чи обчислювальний. На програміста покладається визначення як відбудеться розрізнення координації та обчислень, хоча у більшості випадків таке чітке розділення не носить обов'язкового характеру на синтаксичному рівні координаційної моделі [5].

В основу моделі control-driven координації покладено спільний простір даних, який є асоціативною структурою даних. Спільний простір даних відповідає за зберігання інформації, такої як кортежі. Кортеж є атомарною одиницею зберігання інформації. Не можливо маніпулювати окремим значенням кортежу. Весь кортеж повинен бути отриманий зі спільного простору даних, модифікований і поміщений назад у спільний простір даних. Процес може отримати кортеж за допомогою так званого антикортежу. Спільний простір даних (наприклад,

сервер) порівнює цей антикортеж з наявними кортежами і передає відповідний кортеж початковому процесу.

Отриманий кортеж може бути вилучений з серверу або скопійований. Будь-який процес може помістити процес у спільний простір даних за допомогою примітивів, які підтримуються координаційною моделлю. Крім того, через спільний простір даних реалізовано непрямий зв'язок між процесами. Зміст спільного простору даних не залежить від поточного стану всіх учасників координації, тобто процес може помістити кортеж на сервер та припинити своє виконання, а інший процес може одержати цей кортеж. Тому спільний простір даних також допомагає реалізувати просторове та часове розділення між процесами.

1.2. Модель control-driven координації

Основна різниця між моделями data-driven та control-driven координації полягає у тому, що в останньому випадку існує майже повне розділення координації та власне обчислень. Стан обчислень у будь-який момент часу визначається лише в термінах координаційного шаблону (pattern) до якого належить процес, що бере участь в обчисленнях. Фактичні значення даних, над якими працюють процеси, майже ніколи не беруться до уваги. Стилістично, це означає, що координаційна компонента майже повністю відділена від обчислювальної. Це зазвичай досягається шляхом розробки цілком нової координаційної мови, де обчислювальні частини розглядаються як «чорні скриньки» з чітко визначеними вхідними/вихідними інтерфейсами. Отож, на відміну від data-driven координації, де існує набір наперед визначених примітивів, control-driven координація майже завжди закінчується створенням повноцінної мови. Це також означає, що координаційна control-driven модель стимулює розробника розбивати процеси на дві окремі групи, а саме чисто обчислювальні і суто координаційні процеси.

Принцип роботи control-driven координації ґрунтується на взаємодії постачальника (producer) та споживача (consumer) даних. Ця взаємодія визначається як потік або канал даних між вихідними портами постачальників і вхідними портами споживачів. Такі відносини можна уявити у вигляді конвеєра, де вихідні дані постачальників є вхідними даними споживача. Більше того, процес може пересилати у своє навколишнє середовище подійні або контрольні повідомлення для інформування інших процесів щодо зміни свого стану або запити про стан інших процесів [7].

Такого типу взаємини між постачальниками та споживачами часто призводять до кількох видів залежностей [8]. Першою і найбільш поширеною залежністю між постачальником та споживачем є те, що постачальник повинен завершити свою роботу перш ніж споживач зможе почати працювати. Такого виду залежність називається «обмеженнями передумови» (prerequisite constraints). Для таких залежностей має велике значення повідомлення про те, що процес-споживач може почати роботу. Більш того, необхідні послідовне виконання процесів і наявність окремих процесів, які стежать за тим, що процес-постачальник був завершений. Другий вид залежності називається «передача» (transfer) і відповідає за передачу даних від постачальника до споживача. В даному випадку дані, що передаються – це інформація від постачальника до споживача, і таким чином, передачу даних також можна вважати комунікацією. Третя і остання залежність називається «застосовність» (usability). Ця залежність описує той факт, що результатом роботи постачальника зможе скористатися споживач. Для цієї залежності існує два способи розв'язання: використання запитів до споживача про потрібну йому інформацію чи стандартизація вхідних та вихідних даних.

Яскравим представником control-driven моделі координації є мова програмування Manifold [5].

2. ІМПЛЕМЕНТАЦІЇ LINDA

Окрім класичної Linda, існує багато її розширень та різних варіантів імплементації. Проаналізуємо деякі з них.

2.1. Bauhaus Linda

Bauhaus Linda є безпосереднім продовженням класичної моделі Linda з реалізацією складених просторів кортежів у формі мультимножин (multisets, msets) [5]. Bauhaus Linda не розрізняє кортежі і простори кортежів, кортежі і антикортежі, активні і пасивні кортежі. Замість додавання, читання або видалення кортежів з одного «плоского» простору кортежів, ця реалізація Linda використовує операції out, rd та in, щоб додавати, видаляти, читати мультимножини з іншої мультимножини. Як наслідок, механізм асоціативного пошуку відповідностей Linda, заснований на порядку та позиції елементів у кортежі, замінюється на невпорядковане включення до множин. До того ж ця мова представляє новий примітив move, який використовується для переміщення кортежів вгору чи вниз за рівнями мультимножини, додатковими варіантами якого є операції up та down. Таким чином можна організувати дані в ієрархічні структури.

2.2. Bonita

Bonita описана в [9] Вудом та Роустроном. Ця праця пропонує нові примітиви як з метою покращення функціональності моделі, так і для збільшення продуктивності. Перша мета досягається за допомогою впровадження складених просторів кортежів та агрегованої маніпуляції кортежами. Для досягнення другої цілі водиться більш точний запис отримання кортежу, коли запит від якогось процесу на знаходження кортежу обробляється окремо від перевірки чи отримав процес очікувані дані. Таким чином, ці дії можуть бути виконані паралельно і як наслідок зменшуються накладні витрати.

2.3. JavaSpaces та GigaSpaces

Про JavaSpaces йде мова в [10]. Це імплементація Linda для Java від Sun Microsystems і частина проекту Jini. Об'єктний простір у цій реалізації використовується як простір кортежів Linda, а об'єкти виступають кортежами.

Для роботи JavaSpaces необхідні такі сервіси: вебсервер, RMI activation server, сервіс Jini lookup або сервіс реєстру RMI, менеджер транзакцій Jini, сервер JavaSpaces.

JavaSpaces використовує нестандартну серіалізацію об'єктів для передачі через мережу, при якій серіалізуються лише публічні поля класів. Порівняння об'єктів відбувається побайтно, а не за допомогою методу equals.

GigaSpaces, також згадана у [10] – комерційна імплементація JavaSpaces з використанням нових можливостей, таких як складені кортежі, чи можливість ітерації по кортежах, які відповідають антикортежу.

ВИСНОВКИ

На сьогодні використання складних паралельних розподілених систем є тим шляхом, по якому пішов науковий світ у спробі знаходження розв'язку складних і масштабних обчислювальних проблем. Тому постає актуальне питання про створення ефективних методів розпаралелення і координації окремих обчислювальних задач.

У роботі проаналізовані дві основні моделі координації: data-driven (стан обчислень на будь-який момент часу визначається даними, що отримуються або передаються та фактичною конфігурацію скоординованих компонентів) та control-driven (в основу координації покладено спільний простір даних, який є асоціативною структурою даних).

Крім виділених стилістичних відмінностей між цими координаційними моделями, які впливають на ступінь розділення обчислювальних та координаційних частин, зазначено і різне застосування.

Модель data-driven координації використовується в основному для розпаралелювання обчислювальних задач. Підхід control-driven координації зазвичай використовується для моделювання систем. Це може бути пояснено тим фактом, що в рамках конфігураційного компонента програміст має більший контроль над даними в разі використання мов, що

підтримують data-driven координацію, ніж у випадку control-driven координації. Таким чином, представники першої категорії, як правило, намагаються координувати дані, у той час представники другої намагаються координувати сутності (які, можуть бути не тільки звичайними процесами, а й пристроями, компонентами системи і т. д.).

Короткий опис координаційної класичної моделі Linda та її імплементацій, показує простоту привнесення координаційних механізмів до будь-якої мови програмування.

ЛІТЕРАТУРА

1. Перевозчикова О.Л. Основи системного аналізу об'єктів і процесів комп'ютеризації: – К.: Вид. дім «КМ Академія», 2003. – 432 с.
2. http://folding.stanford.edu/.
3. http://www.mcs.anl.gov/research/projects/mpi/.
4. [Gelernter 1985] David Gelernter, Generative communication in Linda. ACM Transactions on Programming Languages and Systems (TOPLAS), P. 80-112 Volume 7, Issue 1 (January 1985), ISSN:0164-0925 http://www.ece.rutgers.edu/~parashar/Classes/03-04/ece572/papers/gencommlinda.pdf.
5. [Papadopoulos 1991] Coordination models and languages. George A. Papadopoulos, Department of Computer Science University of Cyprus, Nicosia, Cyprus; Farhad Arbab, Department of Software Engineering CWI Amsterdam, The Netherlands, 1991. – http://www.cse.msu.edu/~stire/cse891s04/Papadopoulos.pdf.
6. [Gelernter 1992] David Gelernter, Nicholas Carriereo. Coordination Languages and their Significance, – Communication of the ACM, 1992 (February), Vol. 35, No. 2 http://www.caip.rutgers.edu/~virajb/readinglist/coordinationlang.pdf.
7. Михалевич В.С., Капитонова Ю.В., Летичевский А.А. О методах организации макроконвейерных вычислений // Кибернетика. –1986. – № 3. – С. 3-10.
8. [Malone 1994] Malone T.W. and Crowston K. The Interdisciplinary Study of Coordination. – ACM Computing Surveys 26, 1994 http://ccs.mit.edu/papers/CCSWP157.html.
9. [Rowstron 1996] A. Rowstron and A. Wood. Solving the Linda multiple rd problem. In P. Ciancarini and C. Hankin, editors, *Coordination Languages and Model*, volume 1061 of *Lecture Notes in Computer Science*, April, 1996. http://www.research.microsoft.com/~antr/papers/coord.ps.gz.
10. [Wells 2003] Linda implementations in Java for concurrent system. G.C.Wells from Rhodes University, A.G. Chalmers from University of Bristol and P.G. Clayton from Rhodes University. – Concurrency : Practice and Expertise, 2003; 00:1-7. http://www.cs.bris.ac.uk/Publications/Papers/2000380.pdf.

Стаття надійшла до редколегії 13.04.10 р.