УДК 004.2:004.5

Коваль Ю.В., асистент

Протокол багатоточкового потоку: доведення коректності методом інерційного моделювання

Київський національний університет імені Тараса Шевченка, 83000, м. Київ, пр-т. Глушкова 4д,
e-mail: kafedraTK@unicyb.kiev.ua

Iu. V. Koval, Assistant

Multipoint stream protocol: proof of correctness by the insertion programming method

Taras Shevchenko National University of Kyiv, 83000, Kyiv, Glushkov ave., 4d,

e-mail: kafedraTK@unicyb.kiev.ua

В статті побудовано формалізацію агентів та систему правил що утворюють функцію занурення для вказаних агентів що здійснюють передачу повідомлень за протоколом багатоточкового потоку. Розглянуто занурення одного, двох та трьох агентів в середовище комп'ютерної мережі. Наведено результати моделювання поведінок агентів в середовищі за допомогою системи інсерційного програмування IMS. В статті визначено коректність протоколу багатоточкового потоку та наведено доведення коректності.

Ключові слова: протокол багатоточкового потоку, доведення коректності.

Insertion programming system IMS selected to investigate and prove correctness of multipoint stream protocol. Insertion model for transmitter and receiver agents that communicate through multipoint stream protocol presented and analyzed in the article. Syntax of IMS system is selected to present equation system for transmitter and receiver agents and insertion function. Insertion function for transmitter and receiver agents inserted into network environment built and presented in the article. Examples of multipoint stream protocol usage for one, two and three agents in network environment are constructed and investigated. Total amount of visited state presented. In article is shown that minimum transmitter agent velocity is not less then three times than normal transmitter agent velocity. The possibility of transmitter agents blocking checked. The receiver agent behavior investigated and conclusion is made that no one is blocking other. The correctness of multipoint stream protocol defined. Proof of such correctness proposed.

Key Words multipoint stream protocol, proof of correctness.

Статтю представив д.ф.-м.н., проф. Анісімов А.В.

1. Вступ та постановка задачі

Сучасні методи програмування все більше і більше використовують підходи, пов'язані з вживанням паралельних обчислень. Паралельно виконуються як частини алгоритмів, так і цілі алгоритми. Задача, що вирішується в такі способи має бути розділеною на підзадачі, а загальне рішення або будується із часткових рішень, або є рішенням одної з паралельних гілок обчислень. Другий випадок, як правило, означає розподіл простору, в якому шукають відповідь, на частини та існування повного рішення в одній з таких частин. Конвеєрні обчислення (як специфічний випадок паралельних обчислень) дозволяють готувати значення для обробки наперед (тобто паралельно з обробкою попередніх значень), що також прискорює обчислення в цілому. Всі вказані методи породжують надлишкові обчислення у порівнянні з відповідним базовим алгоритмом, проте кількість цих надлишкових обчислень не повинна бути такою, щоб перекривати виграш у часі за рахунок розпаралелювання обчислень, інакше втрачається сенс самого підходу розпаралелювання. Де факто, час та кількість роботи для виконання базового алгоритму дорівнює сумі часу та кількості робіт паралельних складових (за виключенням вже вказаної невеликої кількості надлишкових організаційних

обчислень). Такий підхід виправданий за умови обмеженого обчислювального ресурсу, що, власне, і було причиною саме такого підходу до паралельних обчислень. Втім, на сьогодні сумарна кількість обчислювальної потужності у світі така, що завантажити її повністю вже неможливо (згадайте скільки часу ви не використовуєте свій десктоп, ноут(нет,ультра)бук, планшет, смартфон та подібні їм пристрої). Ідеї розподілених (астрономічних) обчислень з використанням комп'ютерів поки їх власники їх не використовують (1,2) безумовно були спрямовані на підвищення відсотку утилізації сумарної обчислювальної потужності, проте і вони не в змозі на сьогодні завантажити всі обчислюючи пристрої. Теза про неможливість 100 відсоткової утилізації обчислювального ресурсу може бути підсилена до тези по неминучість зменшення цього показника щонайменше до 20 відсотків у відповідності до принципу Парето (3). Проте і 20 відсотків утилізації здаються видатним досягненням. З іншого боку, ці твердження співзвучні твердженням про збільшення кількості значень на представлення інформаційної одиниці (4) та відповідно кількості обчислень, що потрібно виконати для обчислення інформаційної одиниці. Як результат таких обставин сформулюємо таку тезу: кількість обчислень при паралельному конкурентному обчисленні результату повинна бути (суттєво) більше кількості обчислень будь якого базового алгоритму для вирішення задачі. Це означає наступне.

Нехай для вирішення деякої задачі існує k алгоритмів. Тоді рішення задачі знаходиться методом паралельного запуску всіх таких алгоритмів та отримання результату від найшвидшого у кожному окремому випадку.

Розглянемо це на прикладі задачі сортування масиву. Існують такі алгоритми:

1. бульбашка
2. вибору
3. шейкерне (сортування трусом)
4. Шела
5. пірамідою
6. швидке

Кожен з алгоритмів має свої переваги та недоліки. Проте жоден з них не отримає результат першим на всіх вхідних значеннях. Тому запуск всіх алгоритмів одночасно дає можливість для кожного набору вхідних значень отримати результат за найкращий час. Такий алгоритм обчислення будемо називати конкурентним алгоритмом.

Для технічної реалізації конкурентного алгоритму потрібно середовище, що здатне одночасно передати значення на обробку багатьом обчислювачам. Серед сучасних систем це є комп'ютерні мережі з технологією broadcast (гучномовність). В (5) запропонований протокол багатоточкового потоку, що виконує поставлену задачу. Із усього сказаного випливає наступна постановка задачі: довести коректність вказаного протоколу.

2. Метод вирішення задачі.

В якості методу вирішення задачі оберемо наступний: формалізувати протокол в термінах інерційного моделювання (6) та довести його коректність методами останнього. Також будемо виконувати вказану методику для окремих спрощених випадків застосування протоколу.

Відповідно до загальної схеми роботи протоколу багатоточкового потоку (4) процес, що передає повідомлення за цим протоколом перебуває в одному з наступних станів: St –початковий, T – надсилання, Js – обробка запиту приєднання, Jt - приєднання отримувача. Процес, що отримує повідомлення за цим протоколом, перебуває в одному з таких станів: Sr –початковий, L – очікування, R – отримування. Подіями, що спостерігаються, та складають поведінку середовища та процесів при реалізації з'єднання за протоколом багатоточкового потоку, є виклики методів з'єднання а також відправка та надходження мережевих пакетів. Середовищем, в якому відбуваються події, є мережа передачі та обробки повідомлень. В даному випадку комп'ютерна мережа та комп'ютери, що під'єднані до неї. Відповідно до описаного вище, абеткою подій буде наступна множина: listen, read (connect), read (join), read (data), read (close), write (connect), write (data), write (join), write

(close), TO. Слова read та write позначають дії, що виконують агенти (з відповідним змістом). Слова connect, join, close та data означають повідомлення, якими обмінюються агенти. TO та listen внутрішні дії агентів, що виконуються кожним агентом асинхронно та мають наступний зміст:

- TO – time out – перевищення часу очікування інших подій. Значення часу очікування встановлюється програмістом та залежить від задачі, що вирішується.
- listen – дія, що означає що отримувач готовий приєднатися до багатоточкового потоку.

На малюнку рис. 1 подано графічну схему роботи передавача.

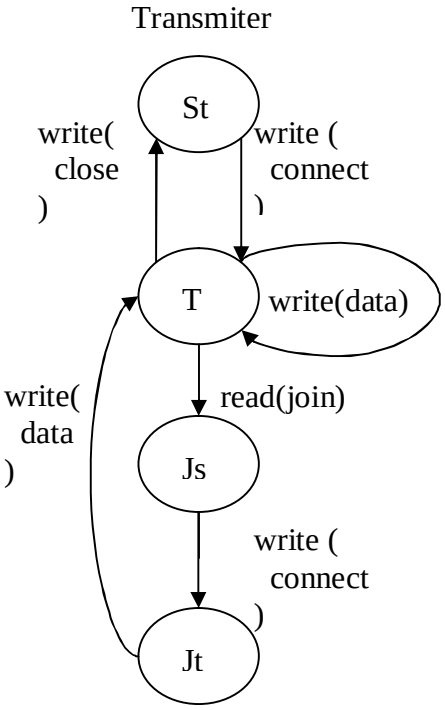

Рис 1. Схема передавача

Передавач одразу знаходяться в стані St який для першого є початковим.

Реалізація абетки зроблена відповідно до наступного сценарію використання протоколу: отримувачі формують окремі точки підключення до потоку та очікують на утворення потоку від передавача. Передавач надсилає повідомлення про створення потоку. Утворюється потік. Передавач надсилає повідомлення. Отримувачі отримують ці повідомлення. Для завершення роботи потоку (якщо таке взагалі передбачено) передавач надсилає повідомлення close. Передавач та отримувачі закривають з'єднання. У випадку нескінченої інформації передача та прийом в потоці продовжуються до нескінченності.

На малюнку рис. 2 подано графічну схеми роботи отримувача.

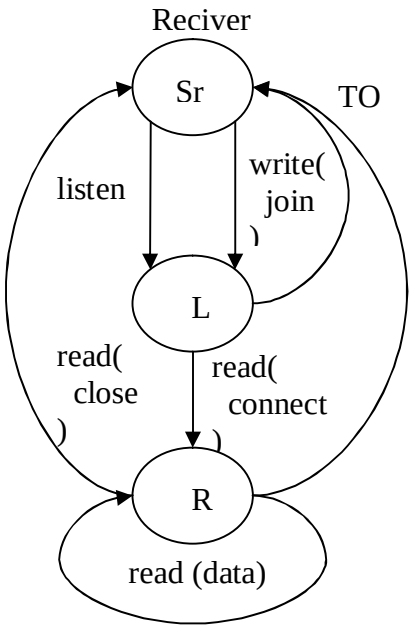

Рис 2. Схема отримувача

Отримувач одразу знаходяться в стані Sr який для першого є початковим.

В термінах інсерційного програмування обидва різновиди процесів іменуються агентами. Агент першого типу (передавач) в одному з'єднанні має бути тільки один. Агентів другого типу (отримувачі) може бути нуль (вироджений випадок) або більше. Середовище, в якому відбуваються всі події, є мережа, що завжди працює, тому вона моделюється агентом Delta. Тоді агенти описується наступними системами рівнянь:

Передавач:

- St = write(connect).T
- T = write(data).T + write(close).St + read(join).Js
- Js = write(connect).Jt
- Jt = write(data).T

Отримувач:

- Sr = listen.L+write(join).L
- L = read(connect).R+TO.Sr
- R =read(data).R + read(close).Sr+TO.Sr

Функція занурення у випадку тільки одного передавача задається наступною системою правил:

- E[write(connect).P]->write(connect).E[P]
- E[write(data).P]->write(data).E[P]
- E[write(close).P]->write(close).E[P]

Всі решта переходів є перехід в 0.

Функція занурення у випадку тільки одного отримувача задається наступною системою правил:

- E[write(join).P]->write(join).E[P]
- E[TO.P]->TO.E[P]
- E[listen.P]->listen.E[P]

Всі решта переходів є перехід в 0.

E, P, a в подальших формулах і Q та R є змінними значеннями яких є стани агентів та середовища. Функція занурення у випадку одного передавача та одного отримувача задається наступною системою правил:

- E[write(connect).P,read(connect).Q] ->transmit(connect).E[P,Q]
- E[write(connect).P,Q] ->write(connect).E[P,Q]| Q!=read(connect).Q’
- E[write(data).P,read(data).Q] ->transmit(data).E[P,Q]
- E[write(data).P,Q] ->write(data).E[P,Q] | Q!=read(connect).Q’
- E[write(close).P,read(close).Q] ->transmit(close).E[P,Q]
- E[write(close).P,Q] ->write(close).E[P,Q] | Q!=read(connect).Q’
- E[read(join).P,write(join).Q] ->transmit(join).E[P,Q]
- E[P,write(join).Q] ->write(join).E[P,Q] | P!=read(join).P’
- E[P,listen.Q]->listen.E[P,Q]
- E[P,TO.Q]->TO.E[P,Q]

Всі решта переходів є перехід в 0.

Подія transmit(x) є одночасна композиція двох дій read(x) одного агента та write(x) іншого агента і є скороченням запису read(x)><write(x).

Кожне правило (наприклад перше) може бути переписано у такому вигляді:

$$\frac{P \xrightarrow{write(connect)} P', Q \xrightarrow{read(connect)} Q'}{E[P,Q] \xrightarrow{transmit(connect)} E[P',Q']}$$

Вжитий раніше синтаксис застосовується в системі інсерційного програмування IMS і займає приблизно той самий об’єм тексту. Тому виходячи з наведених міркувань прийнято рішення застосовувати синтаксис системи IMS. Вертикальна риска означає умову за якої правило може бути застосоване.

Функція занурення у випадку одного передавача та двох отримувачів задається наступною системою правил:

- E[write(connect).P, read(connect).Q, read(connect).R]->transmit(connect).E[P,Q,R]
- E[write(connect).P,read(connect).Q,R] ->transmit(connect).E[P,Q,R] | R!=read(connect).R’
- E[write(connect).P,Q,read(connect).R] ->transmit(connect).E[P,Q,R] | Q!=read(connect).Q’
- E[write(connect).P,Q,R] ->write(connect).E[P,Q,R] | Q!=read(connect).Q’ & R!=read(connect).R’
- E[write(data).P,read(data).Q,read(data).R] ->transmit(data).E[P,Q,R]
- E[write(data).P,read(data).Q,R] ->transmit(data).E[P,Q,R] | R!=read(data).R’
- E[write(data).P,Q,read(data).R] ->transmit(data).E[P,Q,R] | Q!=read(data).Q’
- E[write(data).P,Q,R]->write(data).E[P,Q,R] | Q!=read(data).Q’ & R!=read(data).R’
- E[write(close).P,read(close).Q,read(close).R] ->transmit(close).E[P,Q,R]
- E[write(close).P,read(close).Q,R] ->transmit(close).E[P,Q,R] | R!=read(close).R’
- E[write(close).P,Q, read(close).R] ->transmit(close).E[P,Q,R] | Q!=read(close).Q’
- E[write(close).P,Q,R] ->write(close).E[P,Q,R] |

Q!=read(connect).Q' &
R!=read(close).R'

- E[read(join).P,write(join).Q, write(join).R] ->1:transmit(join).E[P,Q, write(join).R]+ 2:transmit(join).E[P, write(join).Q,R]
- E[read(join).P,write(join).Q,R] ->1:transmit(join).E[P,Q,R]
- E[read(join).P,Q,write(join).R] ->2:transmit(join).E[P,Q,R]
- E[P,write(join).Q,R] ->1:write(join).E[P,Q,R] | P!=read(join).P'
- E[P,Q,write(join).R] ->2:write(join).E[P,Q,R] | P!=read(join).P'
- E[P,listen.Q,listen.R]->listen.E[P,Q,R]
- E[P,listen.Q,R]->listen.E[P,Q,R]
- E[P,Q,listen.R]->listen.E[P,Q,R]
- E[P,TO.Q,TO.R]->TO.E[P,Q,R]
- E[P,TO.Q,R]->TO.E[P,Q,R]
- E[P,Q,TO.R]->TO.E[P,Q,R]

Всі решта переходів є перехід в 0.

У випадку передачі повідомлень з передавача, вони зчитуються отримувачами одночасно. Це підтримано на рівні мережевого середовища та описується одночасною композицією. Тому transmit(connect) в першому правилі є скороченням для такого запису: write (connect) >< (1:read(connect), 2:read(connect)). Так само в правилах подібних до першого. В другому правилі та подібних до нього зберігається попереднє пояснення для дії transmit(x).

Передача запитів на приєднання до потоку отримувачами здійснюється по черзі. Мережеве середовище на базовому рівні забезпечує послідовність (не одночасність) передачі. Окрім того всі повідомлення зберігаються в черзі та обробляються. В описі таких дій вказується, який агент виконав цю дію. Мітки 1 та 2 використовуються саме для цього. Всі агенти отримувачі, що знаходяться в стані L одночасно отримують повідомлення connect та переходять до стану R зчитування потоку.

Дії listen випадково можуть відбутися одночасно, проте оскільки це внутрішні дії кожного агента отримувача, то немає потреби розрізняти їх на рівні середовища. Те саме стосується дій TO. Обидві дії не впливають на працездатність агенту передавача та всієї системи агентів в цілому.

Причина, з якої агент передавач міг би перестати передавати повідомлення полягає в можливості виконання необмеженої кількості запитів приєднання (join) до потоку. Для запобігання такої можливості кожна обробка запиту приєднання супроводжується передачею щонайменше одного повідомлення значень.

Доведення останнього твердження полягає в аналіз слів, що є поведінками агента передавача. Кожне таке слово, в якому зустрічається підслово read (join) . write (connect) має наступним символом write (data). Таким чином, можемо підсумувати наступне: у випадку необмеженої кількості запитів на приєднання агент передавач матиме продуктивність не гірше ніж в три рази менше, порівняно з безперешкодною роботою.

Всі наведені системи правил були перевірені за допомогою системи IMS. Для перших двох систем правил система IMS визначила, що використовуються лише по два стани з усіх можливих. В результаті обробки третьої системи правил, що задає функцію занурення для одного передавача та одного отримувача, було досліджено вже 26 станів середовища. Всі стани визначено як досяжні. Всі стани визначено як покриті.

При обробці четвертої системи правил, що задає функцію занурення для одного передавача та двох отримувачів, кількість досліджених станів середовища сягнула 226. Процес дослідження зайняв приблизно одну хвилину машинного часу комп'ютера з процесором i3. Всі стани визначено як досяжні. Всі стани визначено як покриті.

Дослідження систем з кількістю отримувачів більше двох не змінює досяжність та покриття станів середовища, незважаючи на зростання їх кількості. Окрім того, враховуючи що отримувачі утворюють віртуальний процес, завжди можемо звести задачу з кількома (більше двох) отримувачів до задачі з двома отримувачами.

Досліджені системи дозволяють зробити висновок про коректність протоколу багатоточкового потоку. По-перше, агент передавач завжди може передати потік. По-друге агент отримувач завжди може приєднатися до потоку та отримати значення з нього. По-третє агенти отримувачі не блокують один одного при приєднанні до потоку. По-четверте, у разі непередбачуваної зупинки процесу передавача процеси отримувачі завершуються в разі потреби. Завершення агента передавача залежить лише від внутрішньої коректності самого агента.

Вказані чотири властивості визначають коректність протоколу багатоточкового потоку.

3. Висновки та подальші напрямки роботи.

В результаті проведених досліджень була розроблена та досліджена інерційна модель агентів передавача та отримувача в мережевому середовищі що виконують комунікацію за допомогою протоколу багатоточкового потоку. Визначено та доведено коректність протоколу багатоточкового потоку.

Подальші дослідження будуть спрямовані на те, щоб реалізувати запропонований протокол, дослідити практичну реалізацію, реалізувати на його основі модулі розпаралелювача та аналізатора подій системи розпізнавання жестів та жестової мови та дослідити роботу останньої.

Список використаних джерел

1. List of distributed computing projects / Wikipedia, the free encyclopedia – Режим доступу: https://en.wikipedia.org/wiki/List_of_distributed_computing_projects (14.09.2015)
2. Berkeley Open Infrastructure for Network Computing / BIONIC – Режим доступу: https://boinc.berkeley.edu/ (14.09.2015)
3. Pareto principle / Wikipedia, the free encyclopedia – Режим доступу: https://en.wikipedia.org/wiki/Pareto_principle (14.09.2015)
4. *Крак Ю.В.* До розробки інтерактивного інтерфейсу моделювання та розпізнавання жестової інформації / Ю.В. Крак, Ю.В. Коваль, А.С. Тернов // Вісник Київського національного університету імені Тараса Шевченка Серія фізико-математичні науки. – 2014. – №4. - С.175-178.
5. *Коваль Ю.В.* Протокол багатоточкового потоку: означення та застосування в створенні механізмів розпаралелювача та аналізатора подій в системи жестового інтерфейсу / Ю.В. Коваль // Вісник Київського національного університету імені Тараса Шевченка Серія фізико-математичні науки. – 2015. – №2. - С.152-155.
6. *Летичевский А.А.* Инсерционное моделирование / А.А. Летичевский // УСиМ – 2012. – №6. – С.3-14.

References

1. List of distributed computing projects / Wikipedia, the free encyclopedia – Режим доступу: https://en.wikipedia.org/wiki/List_of_distributed_computing_projects (14.09.2015)
2. Berkeley Open Infrastructure for Network Computing / BIONIC – Режим доступу: https://boinc.berkeley.edu/ (14.09.2015)
3. Pareto principle / Wikipedia, the free encyclopedia – Режим доступу: https://en.wikipedia.org/wiki/Pareto_principle (14.09.2015)
4. KRAK IU.V., KOVAL IU.V., TERNOV A.S. (2014) *To creation of interactive interface for sign information modeling and recognition.* Bulletin of Taras Shevchenko National University of Kyiv Series Physics & Mathematics. Vol. 4. p.175-178.
5. KOVAL IU.V. (2015) *Virtual process: definition and application for gestures interface system creation.* Bulletin of Taras Shevchenko National University of Kyiv Series Physics & Mathematics. Vol. 2. p.152-155.
6. LETICHEVSKY A.A. (2012) *Insertion modeling*. CSaM Vol. 6. p.3-14.

Надійшла до редколегії 15.9.2015