УДК 004.002
DOI: 10.15587/2313-8416.2015.44380

ВПЛИВ ВИСОКОПРОДУКТИВНИХ СОКЕТІВ НА ІНТЕНСИВНІСТЬ ОБРОБКИ ДАНИХ

© В. М. Мельник, Н. В. Багнюк, К. В. Мельник

В роботі вивчаються продуктивність і обмеження сокетів за архітектурою віртуального інтерфейсу (CABI), які використовують компонентну основу для надання підтримки в момент виконання додатків з інтенсивною обробкою даних. Результати показують, що при реорганізації звичайних компонентів досягаються значні покращення продуктивності роботи додатків, а робочі характеристики CABI дозволяють ефективніший розподіл даних у вузлах
Ключові слова: *сокети за архітектурою віртуального інтерфейсу, високопродуктивна мережа, сокети, інтенсивна обробка даних, кластери ПК*

Performance and limitations on the Virtual Interface Architecture Sockets (VIAS) are studied, using a component framework designed to provide runtime support for data intensive applications. The experimental results show that by reorganizing certain components, the significant performance applications improvements are achieving and performance characteristics of VIAS allow a more efficient data partitioning on the nodes
Keywords: *Virtual Interface Architecture Sockets, high-performance network, sockets, data intensive computing, PC clusters*

1. Вступ

Немала кількість досліджень, що супроводжують обробку даних з максимальною потужністю, акцентують увагу на розвитку методів створення складних обчислювальних додатків у різних галузях: науці, техніці, медицині та інших. Подібні програми, як правило, працюють в пакетному режимі і можуть генерувати дуже великі обсяги даних. Передові сенсорні технології також дозволяють досягати високої роздільної здатності для багатовимірних блоків даних. В результаті зростає інтерес до розробки додатків, які в інтерактивному режимі досліджують, узагальнюють і аналізують великі об'єми наукових даних [1]. У даній роботі розглядаються подібні додатки, тобто додатки для інтенсивної обробки даних.

2. Постановка проблеми

Будучи побудованими із стандартних апаратних засобів, кластери ПК стають все рентабельнішими і життєздатнішими альтернативами суперкомп'ютерів основного потоку для широкого спектру додатків, у тому числі для додатків, що оперують інтенсивними даними. Складність у підтримці таких ресурсоємних додатків на платформах полягає в тому, що великі обсяги даних повинні ефективно переміщатися між процесорною пам'яттю. Для досягнення високої продуктивності операції переміщення та обробки даних також повинні бути ефективно скоординованими засобами динамічної підтримки. Поряд з вимогами плідного функціонування, такі додатки вимагають високого рівня гарантій в галузі продуктивності, пристосованості до гетерогенних середовищ і наявності ресурсних змін.

Базові структури на основі компонентів [2–4] можуть забезпечити гнучке та ефективне середовище для додатків з інтенсивною обробкою даних на розподілених платформах. У таких базових структурах додаток є розроблений з набору взаємодіючих програмних компонентів, розміщення яких поряд з ресурсами обчислення відіграє важливість в ступені гнучкості та оптимізації продуктивності програм.

Паралельна обробка даних може бути досягнута шляхом виконання декількох копій компонента в околі кластера зберігання і вузлів обробки [2]. Конвеєр – це ще один можливий механізм для покращення продуктивності. У багатьох додатках з інтенсивною обробкою даних набір (об'єм) даних може розподілятися на визначені користувачем субблоки даних, обробка яких може проходити конвеєрно.

Якщо при організації роботи обробка і зв'язок можуть перекриватися, то покращення продуктивності також набуває залежності від блоковості обчислень і розміру повідомлень даних (тобто субблоків даних). Невеликі блоки даних призводять до покращення балансу навантаження і конвеєризації, а багато повідомлень генеруються блоками малих розмірів. Однак блоки більшого розміру зменшують кількість повідомлень і досягають вищої пропускної здатності каналу зв'язку, а значить, ймовірно, створюють дизбаланс навантаження і зменшення конвеєрності.

З появою сучасних міжкомпонентних високошвидкісних комунікацій, таких як Giganet [2], Myrinet [6], Gigabit Ethernet [7], AHB [8] і Quadrics Network [9], "критична ділянка" у організації зв'язку змістилася до програмного забезпечення обміну повідомленнями. Саме критична ділянка досить організовано і швидко досліджувалася науковцями, спричиняючи створення протоколів на рівні користувача з малим часом затримки та високою пропускною здатністю [10–12]. Поряд з цими дослідженнями, кілька галузей взяли на себе ініціативу стандартизації протоколів високої продукти-

вності на рівні користувача, таких як *архітектура віртуального інтерфейсу* (ABI) [10, 13].

Кілька додатків були розроблені на основі базових протоколів ядра, таких як TCP/UDP, що використовують сокетний інтерфейс [14, 15]. Для їх підтримки на базі високопродуктивних протоколів рівня користувача без будь-яких змін в самому додатку, дослідники підійшли до реалізації ряду підходів. Ці підходи включають в себе сокетні сполучення на рівні користувача в напрямку до протоколів високої продуктивності. Додатки, написані з використанням сокетних з'єднань базових протоколів ядра, часто розробляються з урахуванням продуктивності зв'язку в TCP/IP. З іншого боку, засоби високої продуктивності мають різні характеристики в порівнянні з сокетними зв'язками на базі ядра. Це становить основу критичної ділянки продуктивності, яку здатні забезпечити високопродуктивні засоби. Тим не менш, змінивши деякі компоненти для додатка, такі як розмір субблоку, що утворюють єдиний блок даних, дозволить додаткам взяти перевагу над робочими характеристиками високопродуктивних засобів, роблячи їх більш вагомими і адаптованими.

В даному випадку описується ефективність і обмеження такої реалізації, як CABI, на проміжках продуктивної роботи і гнучкості, які вона дозволяє, в контексті базового компонентного підходу, покликаного забезпечити вчасну підтримку виконання для додатків з інтенсивною обробкою даних, такого як механізм розділення загального блоку даних (БРД) [2]. Слід отримати відповіді на такі питання: чи може дозволити високопродуктивний сокет реалізувати масштабний інтерактивний додаток з гарантіями продуктивності до кінцевого споживача; чи може високопродуктивний сокет покращити адаптованість додатків з інтенсивною обробкою даних в гетерогенних середовищах?

Отримані експериментальні результати показують, що шляхом реорганізації окремих компонентів для додатків можна отримати значні покращення в продуктивності, що призводить до високомасштабованості додатків з гарантіями виконання. Це також дозволяє дрібно-структурований баланс навантаження, роблячи таким чином додатки більш адаптованими до гетерогенних середовищ і наявності ресурсних змін.

3. Огляд додатків з інтенсивною обробкою даних

Зі зростанням обчислювальної потужності і ємності дисків продовжує реально зростати потенціал для створення додатків з метою зберігання багатогігабайтних і багатотерабайтних блоків даних. Глибоке розуміння досягається за рахунок застосування поточного аналізу та кодів візуалізації на збережені дані. Наприклад, інтерактивна візуалізація опирається на можливість отримати внутрішні уявлення, дивлячись на складну систему, в той час як аналіз даних так і візуальне дослідження великих блоків даних відіграють найважливішу роль у багатьох галузях наукових досліджень. Мова йде про додатки, які в інтерактивному режимі роблять запити та аналізують великі блоки наукових даних, тобто додатки з інтенсивною обробкою даних. Прикладом додатка з інтенсивною обробкою даних є оцифровування даних з мікроскопу.

Ключовою проблемою є підтримка програмного забезпечення, яке необхідне для збереження, пошуку, і опрацювання оцифрованих слайдів, для забезпечення інтерактивності протягом робочих інтервалів в ході стандартної поведінки фізичного мікроскопу [16, 17]. Основна складність пов'язана з обробкою великих обсягів даних для цифрового зображення, які можуть варіюватися від декількох сотень мегабайт до декількох гігабайт. На базовому рівні системне програмне забезпечення повинно найбільш точно відображати поточну роботу фізичного мікроскопу, включаючи постійне відслідковування змін та маніпуляції, пов'язані зі збільшенням. Обробка запитів клієнтів вимагає проектування даних з високою роздільною здатністю на відповідну область резолюції з відповідним розподілом пікселів для відображення однієї роздільної точки. Розглянемо сервер візуалізації для цифрової мікроскопії. Клієнт до цього сервера може генерувати різні типи запитів. Найбільш поширеними з них є повне оновлення запитів, за допомогою яких подається повністю нове зображення, а також часткове оновлення запиту, за допомогою якого зображення розглядається як злегка зміщене або збільшене. Сервер повинен бути призначений для обробки обох подібних типів запитів.

Обробка даних в додатках, які подають запит і маніпулюють науковим набором даних, часто може бути представлена у вигляді ациклічного грубого потоку у вигляді визначених блоків від одного або декількох джерел даних (наприклад, один або більше наборів даних, які розподілені по системах збереження) до процесорних вузлів клієнта. Перші дані, що представляють інтерес для такого запиту, отримуються з відповідних наборів даних. Потім дані обробляються за допомогою послідовності операцій на вузлах обробки. Наприклад, в додатку цифрової мікроскопії дані, що представляють інтерес, обробляються за допомогою операцій об'єднання субблоків, операцій перегляду [2, 18] і, нарешті, оброблені дані посилаються клієнту.

Дані, що утворюють частини зображення, зберігаються у вигляді блоків або фрагментів з точки зору індексації, необхідної для отримання цілого блоку, навіть у випадку, якщо для цього потрібна тільки частина блоку. На рис. 1 показане повне зображення, що складається з декількох блоків. Як видно з рисунка, при частковому запиті оновлення (прямокутник з пунктирними лініями на кресленні) може знадобитися тільки частина блоку. Тому розмір і протяжність блоку впливає на кількість даних, необхідних для отримання та передавання вищеописаних запитів.

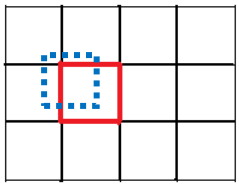


Рис. 1. Представлення розмітки розбиття повного блоку даних на субблоки. Частковий запит, обведений прямокутником з пунктирними лініями, який вимагає лише частину блоку

4. Завдання, пов'язані з підтримкою продуктивності виконання для додатків з інтенсивною обробкою даних

Для додатків з інтенсивною обробкою даних, продуктивність може бути покращена в декількох напрямках. По-перше, набори даних можна декластеризувати у всій системі для удосконалення паралелізму вводу-виводу при їх отриманні, які і представляють інтерес запиту. При відповідності підбору декластеризації запит потрапить до максимально можливої кількості дисків. По-друге, потужність обробки даних системою може бути використана ефективно, якщо додаток спроектований для використання паралелізму при їх обробці. Іншим фактором, який може покращити продуктивність, особливо в інтерактивному дослідженні наборів даних, є конвеєрне виконання їх обробки. Загальний час виконання обробки додатком може бути зменшений при умові, якщо дані розділити на блоки (шматки) і реалізувати конвеєрну їх обробку. У багатьох додатках, конвеєрна обробка також забезпечує механізм, що поступово створює сукупний продукт вихідних даних. Іншими словами, користувачеві не потрібно чекати завершення обробки запиту, так як часткові результати будуть генеруватися поступово. Хоча це не може фактично зменшити загальний час обробки, проте така особливість дуже ефективна в інтерактивних установках, особливо якщо область зацікавлення постійно зміщується.

Робочий розподіл і розмір блоків даних впливають на продуктивність конвеєрного виконання. Розмір блоків повинен бути ретельно підібраний з урахуванням пропускної здатності мережі і часу затримки (час очікування, необхідний для передачі повідомлення, включаючи обробки за допомогою протоколів на всьому проміжку передачі, починаючи від відправника і до кінцевих одержувачів). Зі збільшенням розміру блоку кількість повідомлень, яка необхідна для передачі одного і того ж об'єму даних – зменшується. У цьому випадку, пропускна здатність як характеристика стає важливішою, ніж час затримки. Однак, зі збільшенням розміру блоку час обробки на кожен такий блок також збільшується. В результаті, система стає менш чутливою, тобто, частота окремих/послідовних оновлень зменшується. З іншої сторони, якщо розмір блоку менший – збільшується кількість повідомлень. В результаті час очікування може стати домінуючим фактором у загальній ефективності додатку. Крім того, невеликі фрагменти спричиняють покращення балансу навантаження між копіями компонентів додатка, але затрати на зв'язок можуть компенсувати приріст продуктивності.

Великі блоки даних дозволяють застосовувати кращий час реакції системи для повного оновлення запиту, пов'язаний з покращенням пропускної здатності. Тим не менше, частковий запит на оновлення приведе до збільшення кількості даних і, як результат, затрат на них. З іншого боку, з малим розміром блоку, частковий запит на оновлення не отримуватиме багато непотрібних даних, але повний запит на оновлення буде зазнавати втрат через зниження пропускної здатності.

До того ж, забезпечуючи більш високу пропускну здатність і знижуючи рівень затримки, сокети високої продуктивності мають інші цікаві можливості, показані на рис. 2, *а, б*. Рис. 2, *а* показує, що вони збільшують пропускну здатність при набагато меншому розмірі повідомлення в порівнянні з виміряними на базових kernel-сокетах, таких як TCP/IP. Наприклад, для досягнення пропускної здатності, базовим *kernel*-сокетам потрібен розмір повідомлення U1 байт, в той час як високопродуктивні сокети вимагають меншого розміру повідомлення U2 байт. Використовуючи інформацію, подану на рис. 2, *б* зауважимо, що вони призводять до зниження часу затримки повідомлення, понижуючи його від L1 до L2 при сталому розмірі повідомлення U1 в байтах. Також виявлено, що високопродуктивні сокети при використанні розміру повідомлення U2 байт (рис. 2, *а*) надалі скорочують час затримки від L2 до L3 і, в результаті, спостерігається підвищення продуктивності.

Неоднорідність виникає в декількох ситуаціях. По-перше, апаратне середовище може складатися з машин з різної потужності роботи і ємності пам'яті. По-друге, ресурси можуть розподілятися між іншими додатками. В результаті, наявність ресурсів, таких як CPU і пам'ять може динамічно змінюватися. У таких випадках, додаток повинен бути структурно продуманим і пристосованим до гетерогенної природи середовища при організації його роботи. Він також повинен бути оптимізованим з точки зору використання спільних ресурсів і адаптованим до змін їх наявності та доступності до них. Це вимагає від додатка застосування механізмів адаптації, щоб збалансовувати робоче навантаження між вузлами обробки в залежності від обчислювальних можливостей кожного з них. Для можливого вирішення цього питання здійснюється планування заздалегідь вихідних даних і розрахунків між вузлами обробки. Дані можуть бути розбиті на фрагменти так, щоб це могло дозволяти здійснювати і обробку даних і комунікацію в конвеєрнму режимі.

До того ж, призначення фрагментації даних для вузлів обробки можна зробити з урахуванням схеми попиту, так щоб швидші вузли отримали більше даних для обробки. Якщо швидкодіючий вузол сповільнюється (наприклад, через дії інших додатків), то механізм, що лежить в основі балансування навантаження, повинен бути здатним швидко виявляти зміну наявності та доступності до ресурсів.

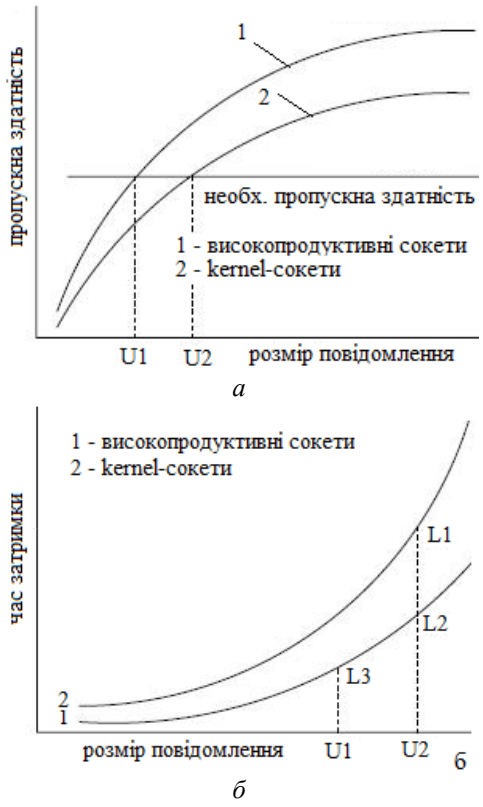


Рис. 2. *а* – Досягнення заданої пропускної здатності проміжними високопродуктивними сокетами для меншого розміру повідомлення в порівнянні з kernel-сокетами; *б* – можливість досягнення прямого і непрямого покращення продуктивності високопродуктивними сокетами, обумовлених характеристиками додатків

5. Інфраструктура програмного забезпечення для надання оцінки

Під час розробки додатків і підтримки їх роботи компонентно-базові засоби [2–4] можуть забезпечити ефективне середовище для вирішення проблем у високопродуктивних додатках. Компоненти можуть бути розміщені на різних обчислювальних ресурсах, а завдання і паралельна обробка даних може досягати вдосконалення шляхом конвеєрного виконання багатьох копій цих компонентів. Таким чином, в даній роботі використовується базова компонентна інфраструктура блокового розбиття даних (БРД) [2], призначена для підтримки додатків з інтенсивною обробкою даних в розподілених середовищах. Також використовується високоефективний інтерфейс сокетів CABI, розроблений для додатків, що використовують TCP/IP, з метою надати переваги в реалізації можливостей продуктивності ABI.

Далі коротко опишемо структуру засобу ділення даних на блоки БРД [2], який реалізує програмну фільтр-потокову модель для розробки додатків з інтенсивною обробкою даних. У цій моделі структура робочого додатока реалізована у вигляді набору компонентів, названих фільтрами, що здійснюють обмін даними через абстрагування (розділення) потоку. Інтерфейс для фільтра складається з трьох функцій:

1) функція ініціалізації, в якій розміщені та визначаються всі необхідні ресурси, такі як пам'ять для структури даних;

2) функція обробки, в якій операції, визначені користувачем, застосовуються на елементах даних;

3) функція завершення, в якій проходить звільнення ресурсів, що задіяні в функції ініціалізації.

Фільтри з'єднані за допомогою логічних потоків. Потік визначається як однонапрямлений потік даних від одного фільтра (відправника розсилань) до іншого (отримувача). Фільтр необхідний для читання даних з його вхідних потоків і запису даних виключно в його вихідні потоки. В роботі визначено буфер даних у вигляді масиву елементів, переданих від одного фільтра до іншого. Оригінальна реалізація логічного потоку доставляє дані в буфери фіксованого розміру і використовує TCP для потокової взаємодії між цими точками.

Загальна структура обробки додатка реалізується за допомогою фільтрової групи, яка являє собою набір з'єднаних між собою фільтрів за допомогою логічних потоків. Коли група фільтрів запускається для обробки запиту додатка, то перш ніж почати виконання запиту програми, робоча система в цей момент здійснює з'єднання сокетів між фільтрами, розміщеними на різних *хост*-комп'ютерах. Фільтри, розміщені на одному комп'ютері, діють як окремі нитки. Запит додатку обробляється групою фільтрів як єдиний робочий модуль (ЄРМ). Прикладом може бути візуалізація набору даних в залежності від кута спостереження. Обробка ЄРМ може бути здійснена в конвеєрному режимі, в якому різні фільтри можуть працювати з різними елементами даних одночасно. Вона починається, коли послуга фільтрації викликає функцію ініціалізації фільтра, який знаходиться там, де можуть бути попередньо розташовані всі необхідні ресурси, наприклад, такі як пам'ять. Далі активується функція обробки, призначена для читання даних з будь-яких вхідних потоків, роботи з даними, що надходять і розміщуються по буферах, і запису даних в будь-які вихідні потоки. Після обробки останнього буфера передається спеціальний маркер від системи обробки, щоб відзначити кінець для поточного ЄРМ (рис. 3, *а*). Функція завершення або фіналізації викликається після завершення обробки поточного ЄРМ з метою звільнення задіяних ресурсів, таких як робочий простір. Функції інтерфейсу можуть активуватися в подальшому з метою обробки наступного ЄРМ.

Модель програмування надає кілька розподілень для полегшення оптимізації продуктивності. Прозора копія фільтра – це копія у групі фільтрів (рис 3, *б*). Вона прозора в тому сенсі, що розподіляє ті ж самі логічні вхідні і вихідні потоки вихідного фільтра. Прозора копія фільтра може бути зроблена, якщо відсутній вплив семантик фільтрованої групи. Тобто, вихідний вигляд робочого блоку повинен бути незмінним, незалежно від числа прозорих копій. Прозорі копії дозволяють паралелізм даних тільки для виконання одиничного запиту, в той час як кілька груп фільтрів дозволяють паралелізм між декількома запитами.

Фільтрова система в режимі виконання підтримує ідею єдиного логічного потоку від точки до точки для організації зв'язку між логічним фільтром-відправником і логічним фільтром-отримувачем. Він відповідає за елементи планування (або буфери) в потоці даних між прозорими копіями фільтра. Для прикладу, на рис. 3, *б*, якщо копія P1 видає операцію запису буфера в логічній потік, який виконує з'єднання від фільтра P до фільтра F, буфер може бути відправлений у вигляді копій в host3 або host4. Для розподілу між прозорими примірниками система виконання підтримує циклічно мігруючий механізм (ЦММ) і механізм керування попитом (МКП) на основі рівня буфера споживання. МКП спрямований на відправку буферів на фільтр, що найшвидше мав би обробляти їх. Коли фільтр споживача починає обробку буфера, отриманого від фільтра-відправника, він посилає йому повідомлення підтвердження, щоб відмітити, що буфер знаходиться в процесі обробки. Фільтр-відправник вибирає споживчий фільтр з мінімальною кількістю непідтверджених буферів для відправки даних буферу і, таким чином, досягаючи кращого балансу між навантаженням.

Незважаючи на розвиток протоколів з малим часом затримки і високою пропускною здатністю на рівні користувача, багато додатків були розроблені раніше на основі *kernel*-протоколів, таких як TCP і UDP, а деякі з цих додатків потребували досить багато часу для їх розробки. Спроби переписати ці додатки на основі протоколів рівня користувача є досить трудомісткими і непрактичними. З іншого боку, інтерфейс сокетів широко використовується в різноманітних додатках, написаних на основі протоколів, таких як TCP і UDP. Мережа cLAN є апаратним втіленням віртуального інтерфейсу ABI (архітектури віртуального інтерфейсу або в літературі – Virtual Interface Architecture (VIA)). Є дві типові сокетні реалізації у мережі cLAN. Одна з них є для утримання сокета наслідування, TCP/UDP і IP рівнів незмінними, у той час як один додатковий рівень вводиться для з'єднання *IP*-рівня і *kernel*-рівня рівня віртуального інтерфейсу. Застосування LANE (LAN Емулятора) на рівні сокетів – це така реалізація, яка використовує рівні від IP до BI [10]. Завдяки вершинам системних викликів (в тому числі контекстний *kernel*-перемикч, заповнення кешу, обробка TLB, обробники пристроїв підрівнів і т.д.) і мультикопіям, введеним в цю реалізацію, додатки, що використовують LANE не були в змозі досягти повної переваги високої продуктивності, представленої в базовій мережі. Інший тип сокетів в мережі cLAN реалізований для забезпечення сокетного інтерфейсу з використанням бібліотек рівня користувача, що базуються на ABI-примітивах рівня користувача. Реалізація в даній роботі потрапляє в цю категорію. Іде посилання на визначений сокетний рівень, такий як CABI, в решті частини викладеного матеріалу. Так як реалізація CABI не є основною в даній статті, то результати мікро-тестів для нашого рівня сокетів будуть представлені в наступному розділі. Для інших деталей, пов'язаних з розробкою та здійсненням CABI, можна звернутися до роботи [20].

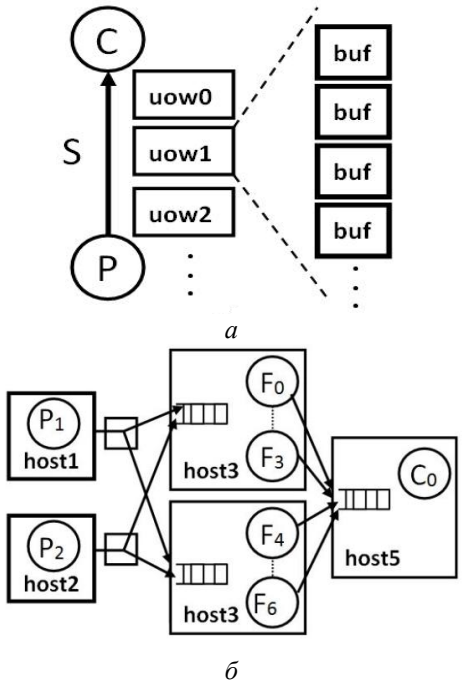


Рис. 3. БРД в потоковому абстрагуванні і підтримка копій: *а* – розміщення буферів даних і маркерів кінця роботи потоку; *б* – встановлення груп фільтрів P, F, C при використанні прозорих копій

6. Результати дослідження

Оцінка ефективності

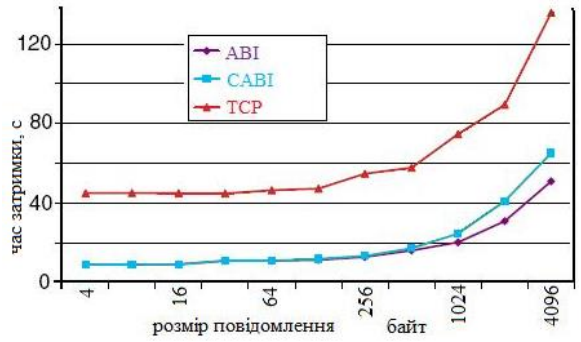
Далі буде представлено дві групи результатів. По-перше, буде досліджуватись максимальна продуктивність, надана через CABI у вигляді мініетестів на час затримки і пропускну здатність. По-друге, буде розглянуто прямий і непрямий вплив на продуктивність, представлену високопродуктивними сокетами в додатках, реалізованих з використанням БРД для того, щоб оцінити обидві характеристики: час затримки і аспекти пропускної здатності на шляху контролю.

Експерименти проводилися на кластері ПК, який складається з (16 виходів) 420 вузлів, з'єднаних мережами Giganet cLAN і Fast Ethernet. Використовуються хост-адаптери cLAN 1000 і кластерні свічі cLAN5300. Кожен вузол має два процесори частоти 1 ГГц типу Pentium III, побудованих навколо Intel 840 чіпсет, який має чотири 32-бітних 33 МГц PCI роз'єми. Ці вузли оснащені 512 Мбайт SDRAM і 256 К кеш-пам'яттю рівня L2. Версія ядра Linux-2.2.17.

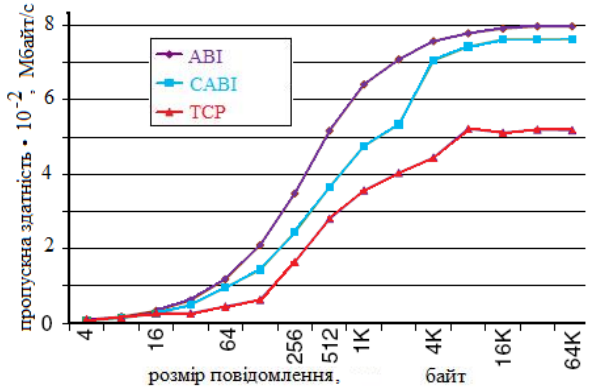
Рис. 4, *а* демонструє час затримки, досягнутий високопродуктивним сокетом в порівнянні з результатом, що досягається за допомогою традиційної реалізації сокетів на вершині TCP і безпосереднього застосування VIA (базової VIA). Реалізований сокетний рівень дає низьку затримку – 9,5 с, що дуже близька до тієї, що надає VIA. Крім того, спостерігається покращення майже в п'ять разів в порівнянні з часом затримки, що задається на рівні традиційних сокетів через TCP/IP.

Рис. 4, *б* показує пропускну здатність, досягнуту високопродуктивним сокетом в порівнянні з використанням традиційних сокетів при базовому

використанні cLAN VIA. CABI досягає пікової пропускної здатності 763 Мбайт/с в порівнянні з 795 Мбайт/с, наданої VIA і 510 Мбайт/с, що надаються традиційною реалізацією TCP. Спостерігається покращення майже на 50 %.



а



б

Рис. 4. Порівняльні графіки: *а* – часу затримки; *б* – пропускної здатності

Продуктивність і поведінка додатку

У цих експериментах використовувалися два види додатків. Перший додаток емулював сервер візуалізації. Це програма, що використовує чотирьохступінчатий конвеєр з залученим фільтром візуалізації на останньому етапі. Крім того, проводились три копії кожного фільтра в конвеєрі, з метою покращення кінцевої пропускнної здатності (рис. 5). Користувач візуалізує зображення у вузлі візуалізації, на якому знаходиться фільтр візуалізації. Необхідні дані витягуються зі сховища даних і передаються на інші фільтри, кожен з яких розміщений на іншому вузлі в системі, що охоплюється конвеєром.

Кожне зображення, що розглядається користувачем, вимагає даних об'ємом в 16 Мбайт, які отримуються і обробляються. Ці дані зберігаються у вигляді фрагментів заздалегідь визначеного розміру, які називають розміром блоку розподілу. Для типового розміру блоку розподілу повний образ складається з декількох блоків (рис. 1). Коли користувач запрошує оновлення для зображення (часткового або повного), відповідні блоки повинні бути доставлені і включені в єдине зображення. Кожен блок витягується цілісно, що потенційно може призводити до активації додаткових непотрібних даних, які можуть бути передані по мережі.

В роботі були змодельовані два види запитів. Перший запит являє собою повне оновлення або запит на нове зображення. Він вимагає, щоб всі блоки, що відносяться до запиту, були інтегровані в одне зображення. Цей вид оновлення є чутливим до пропускної здатності, тому було б добре використовувати великий розмір блоку. Таким чином, як описано вище, для звичайного рівня оновлення запитів з певною швидкістю повинен бути використаний звичайний визначений розмір блоку (або більший).

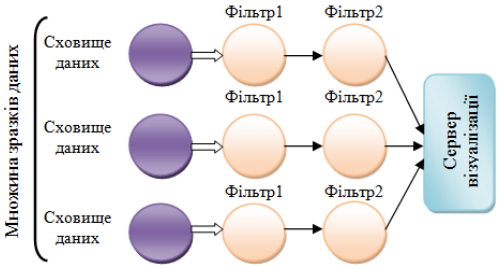


Рис. 5. Оцінка ефективності на основі гарантії – експериментальна установка

Другий запит реалізував часткове оновлення. Цей тип запиту виконується, коли користувач частково зміщує вікно візуалізації, або намагається змінити масштаб зображення, що розглядається в даний час. Для часткового оновлення запиту потрібно тільки додати надлишкові блоки, яких, як правило, невелика кількість у порівнянні з кількістю блоків, що утворюють повне зображення. Цей вид оновлення є чутливим до часу затримки. Крім того, фрагменти в цілому відновлюються (витягуються або звужуються). Таким чином, було б корисно мати невеликі блоки.

Якщо розмір блоку досить великий, часткове оновлення, ймовірно, займе більше часу, так як весь блок видозмінюється, навіть якщо потрібна невелика його частина. Якщо ж розмір блоку занадто малий, повне оновлення займе довший час, так як повинні бути видозмінені багато менших за розміром блоків. Таким чином для додатку, який дозволяє обробку обидвох видів запитів, необхідно встановити продуктивний компроміс між виконанням цих двох типів запитів. У наступних експериментах демонструється покращення масштабованості додатку з VIA-сокетом у порівнянні з TCP та гарантіями виконання для кожного виду оновлення. Результати будуть представлені нижче.

Другий додаток, що розглядається – це механізм балансування навантаження програмного забезпечення, один із таких, що використовується засобом БРД. Коли дані обробляються кількома вузлами, то ідеальна конвеєрність даних досягається тоді, коли час, необхідний для механізму балансування навантаження для того, щоб посилати один блок повідомлення до вузла обробки, дорівнює часу, витраченому вузлом обробки, щоб його обробити. У цьому додатку, типовий розмір блоку вибирається так, щоб досягти досконалої збалансованої конвеєрності між обробкою і комунікацією. Вважається, що потужність обчислення вузлів не змінюється в ході процесу виконання програми. В гетерогенному динамічному середовищі це припущення не підтримується. У наших експериментах в однорідній обстановці досконала узгоджена конвеєрність досягається при

16 Кбайт і при 2 Кбайт для TCP/IP і VIA, відповідно. Це означає, що розмір блоку необхідний для TCP/IP значно більший, ніж для VIA. Однак в гетерогенних мережах, коли розмір блоку помітно великий, помилка в балансуванні навантаження (передача блоку даних до сповільненого вузла) може стати занадто "дорогою" (рис. 6). Вплив на продуктивність з такою неоднорідністю буде розглянутий нижче.

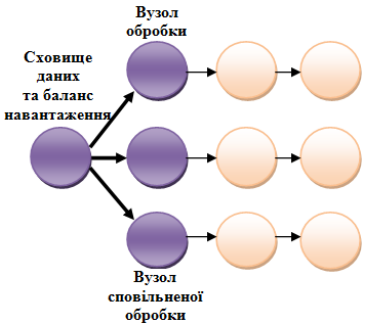


Рис. 6. Експериментальне представлення даних для ефекту гетерогенного кластера

Оцінка ефективності на базі гарантії

Візьмемо до уваги *ефект дослідження середнього часу затримки з гарантією на оновлення за кожну секунду*. У першій серії експериментів користувач намагається удосконалити заданий покадровий рівень (тобто кількість згенерованих нових зображень або повних оновлень, виконаних за секунду). З цим обмеженням досліджується середнє значення часу затримки, що спостерігається при частковому запиті на оновлення. Рис. 7, *а, б* демонструють ефективність роботи, досягнуту додатком. При заданій частоті кадрів для нових зображень TCP вимагає певного розміру повідомлення для досягнення необхідної пропускної здатності. З фрагментацією даних на частини, зробленою відповідно до цієї вимоги, час затримки для часткового оновлення за допомогою TCP повинен би відповідати часу затримки для цього фрагменту повідомлення (позначений як TCP на легенді). З тим же розміром фрагменту розбиття даних CABI забезпечує більш високу продуктивність (позначка CABI на легенді). Проте CABI вимагає набагато меншого розміру повідомлення, щоб досягти пропускної здатності для повних оновлень. Так при перерозділенні даних (ПД) на блоки з урахуванням часу затримки і пропускної здатності для CABI, час затримки може бути додатково зменшений (позначка CABI з ПД на легенді). Рис. 7, *а* демонструє продуктивність без будь-якої обробки. Цей експеримент підкреслює важливу вигоду, отриману за допомогою CABI без впливу наявних обчислювальних затрат на кожному вузлі. Можна спостерігати, що TCP не може досягти обмеження оновлення більшого, ніж 3,25 повних оновлень за секунду. Однак CABI (з позначенням ПД), як і раніше, можуть досягти цього кадрового рівня без помітного погіршення в продуктивності. Результати, отримані в цьому експерименті, показують покращення більш ніж в 3,5 рази без будь-яких змін в блоковому перерозподілі і більш ніж в 10 разів з наявністю блокового перерозподілу даних. Разом з CABI покращення продуктивності роботи додатка з реорганізацією його деяких компонентів (розмір блоку в даному випадку) дозволяє досягти значних переваг не тільки в роботі, але і в масштабованості з гарантіями продуктивності.

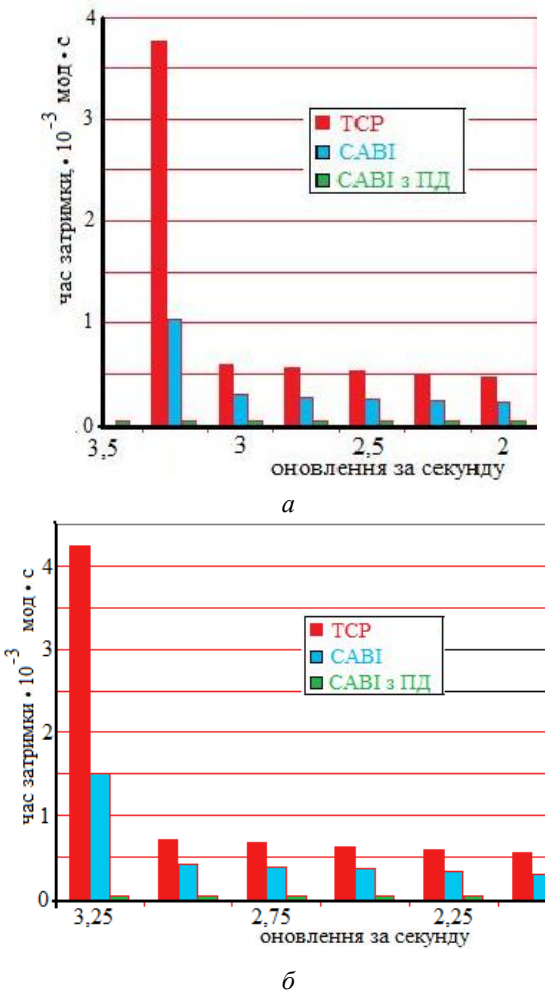


Рис. 7. Вплив високопродуктивних сокетів на середній час затримки з гарантіями оновлень в секунду: *а* – без обчислення вартості; *б* – з лінійним обчисленням вартості

Рис. 7, *б* демонструє роботу додатка з урахуванням затрат на обробку, яка є лінійною від розміру повідомлення в експериментах. Було проведено вимірювання часу обробки, необхідного для візуалізації частини додатка цифрової мікроскопії, названої віртуальним мікроскопом [17], на БРД і виявлено, що це становить 18 nc на один байт повідомлення. Подібні програми, що використовують перегляд цифрованих мікроскопічних слайдів, мають низькі затрати обробки на піксель. Це програми, що отримують перевагу в залежності від низького часу затримки і високої пропускної здатності проміжних вузлів. Таким чином, в цій роботі зосередження формується на подібних додатках.

На відміну від попереднього експерименту, в цьому експерименті навіть CABI (з ПД) не змогли досягти рівня, вищого за 3,2. Причина цього полягає в тому, що пропускна здатність, задана CABI, обмежена

затратами обробки в кожному вузлі. У цьому експерименті спостерігається покращення більш ніж на 4 рази безблокового перерозподілу даних та на 12 разів з блоковим перерозподілом даних, відповідно.

Розглянемо ефект оновлення за секунду з гарантіями часу затримки. У другій серії експериментів іде намагання максимально збільшити кількість повних оновлень в секунду, коли конкретне значення часу затримки є піково-результуючим для часткового запиту на оновлення. Рис. 8, *а, б* демонструють покращення продуктивності, досягнутої додатком. Для конкретно заданого часу затримки TCP не може мати розмір блоку більший, ніж певне значення. З розбиттям даних, відтвореним відповідно до цієї вимоги, покращення пропускної здатності є дуже обмеженим, як видно на малюнку з позначкою "TCP". З таким же розміром блоку CABI досягає набагато кращої продуктивності, що демонструється на рисунку з позначкою "CABI". Однак, перефрагментація даних, що враховує час затримки і пропускну здатність CABI, що згадувались в результатах обговорення вище, досягає набагато вищої продуктивності, як показано на рисунку (зі значенням продуктивності, позначеної "CABI ПД"). Рис. 8, *а* подає значення продуктивності без будь-якої обробки даних, а значення продуктивності з урахуванням затрат на обробку даних, що лінійно залежить від розміру блоку, демонструється в експериментальних результатах на рис. 8, *б*. Без затрат на обробку, коли час затримки обмежується і стає досить низьким, біля 100 μs, TCP різко падає. Однак, CABI продовжує роботу з характеристиками, близькими до пікових значень. Результати експерименту демонструють покращення більше ніж в 6 разів без будь-якого перерозподілу блоків даних, а також більш ніж у 8 разів з застосуванням блокового перерозподілу даних. Враховуючи витрати на обробку даних можна бачити, що з великою гарантією часу затримки робота з TCP і CABI дуже схожа. Причина в цьому полягає в затратах на обробку на шляхах сполучення. З затратою на обробку у 18 пс/байт, обробка даних все-таки виходить на рівень незростання з VIA, а з використанням TCP система передачі скоріше досягає цього рівня. З однакової причини, на відміну від TCP, кадровий рівень, досягнутий CABI, непомітно змінюється зі зменшенням часу затримки. Результати цього експерименту показують підвищення рівня продуктивності до 4 разів.

Далі розглянемо ефект *впливу декількох запитів на середній час зворотньої реакції.* У третій серії експериментів розглянута модель з наявністю двох видів запитів. Перший тип запиту – це запит на масштабування або збільшення, а другий – запит на повне оновлення. Перший запит охоплює невелику область зображення, що вимагають тільки чотири блоки даних, які повинні бути відновлені. Другий запит охоплює все зображення, таким чином всі блоки даних повинні бути відновлені і оброблені. Рис. 9, *а, б* відображають середній час зворотнього відклику на запити. Вісь *x* показує частину запитів, які відповідають другому типу. Решта запитів відповідають першому типу. Об'єм блоку даних, встановлений для кожного запиту, залежить від розбиття повного обсягу даних на блоки. Оскільки частина запитів кожного виду не може бути відома апріорно, то аналізується ефективність TCP і CABI з різними розмірами поділу. Якщо повний набір даних не розділений на блоки, запит має доступ до всіх даних, тому затрати часу не змінюються зі зміною частки запитів. Переважаючий вклад CABI в порівнянні з TCP полягає лише у властивій перевазі CABI і не має нічого спільного з розмірами блоків. Однак з перерозподілом набору даних на більш дрібні блоки, рівень зростання часу відклику стає досить високий для TCP в порівнянні з CABI. Таким чином, для будь-якого середнього фіксованого часу відклику CABI може витримувати більші зміни в частці різних типів запитів, ніж TCP. Наприклад, при середньому часі зворотнього відклику 150 мс і 64 блоки на повний набір даних, TCP може підтримувати зміни в діапазоні до ~60 % (у відсотках від повного оновлення запитів), але після цього спадає. При таких же умовах CABI може підтримувати зміни в діапазоні до ~90 % перед незначним зниженням. Це показує, що в тих випадках, коли розмір блоку не може бути передбачений і наперед визначений, або коли тільки є правильний вибір розміру блоку, CABI діє набагато краще.

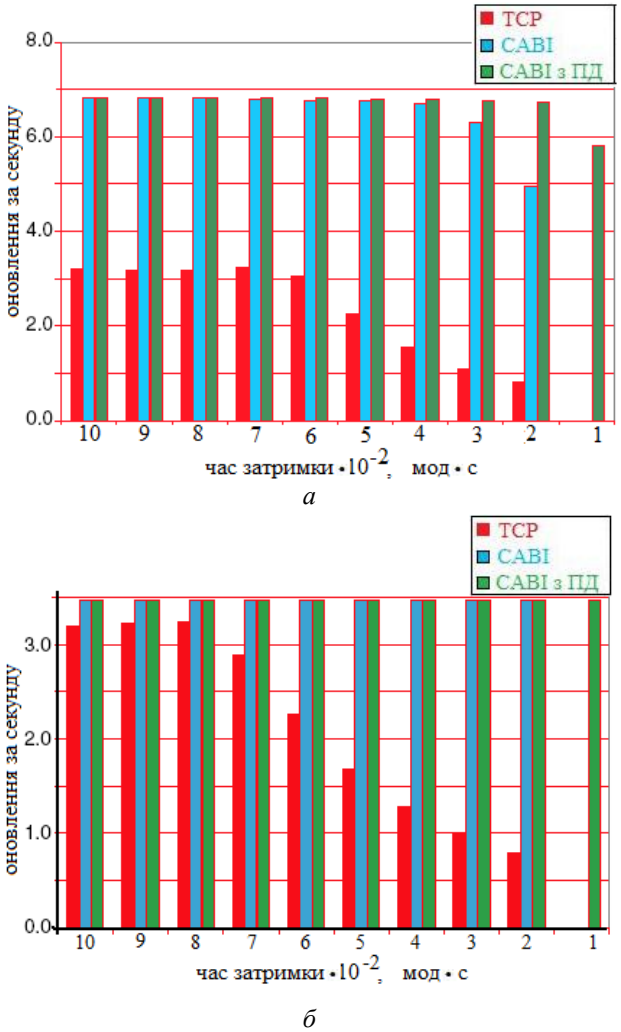


Рис. 8. Ефект впливу високоефективних сокетів на оновлення за секунду з гарантіями часу затримки: *а* – без врахування затрат на обробку; *б* – з лінійними затратами на обробку

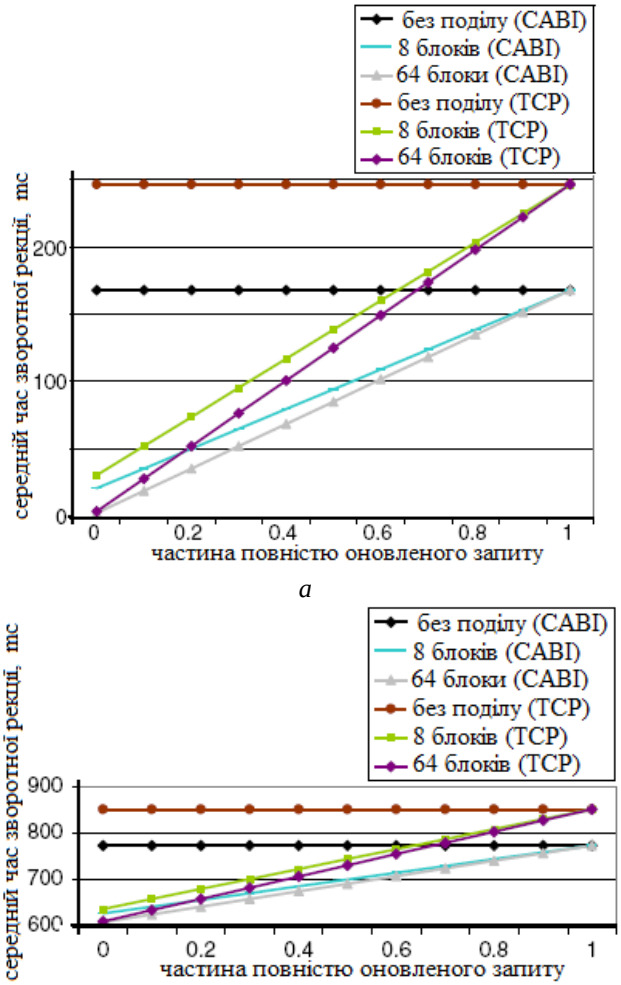
Вплив CABI на гетерогенні кластери

У експериментах аналізується ефект впливу CABI на кластер з набором гетерогенних обчислювальних вузлів. Емулюються повільніші вузли в мережі, змушуючи деякі вузли обробляти дані більш ніж один раз. Для *хост*-протоколів, таких як TCP, зменшення швидкості обробки може призвести до погіршення часу зв'язку, а заодно, до погіршення часу обробки. Все ж, в цих експериментах припускається, що час комунікації залишається сталим, і тільки змінюється час обробки.

Розглянемо *вплив циклічної схеми планування на гетерогенні кластери.* В цьому експерименті розглядається вплив на продуктивність в буфері з ЦММ, запланованому в БРД з використанням TCP і CABI. Для досягнення ідеальної конвеєризації час, необхідний для передачі даних до вузла, повинен дорівнювати часу обробки даних на кожному з вузлів. Для цього експерименту розглядається баланс розподілу навантаження між фільтрами додатка візуалізації (перші вузли в конвеєрній лінії на рис. 6). Час обробки даних в кожному фільтрі є лінійним від розміру повідомлення і становить ~18 nc/байт повідомлення. З участю TCP ідеальну конвеєрість було досягнуто для повідомлень до 16 Кбайт, а для CABI цей показник було досягнуто за допомогою повідомлень в 2 Кбайти. Таким чином, балансування навантаження може охоплювати більшу деталізацію.

Рис. 10 демонструє кількість часу, необхідну для балансування навантаження, необхідного для досягнення гетерогенності вузлів зі збільшенням фактора гетерогенності в мережі. Фактор неоднорідності співвідноситься зі швидкістю обробки найшвидших і найповільніших процесорів. З використанням TCP розмір блоку є великим (16 Кбайт). Таким чином, коли при балансуванні навантаження виникає помилка (блок передається до більш повільного вузла), це призводить у повільному вузлі до затрати значно більшої кількості часу на обробку цього блоку. Це збільшує час, який балансування навантаження потребує для реалізації його помилки. З іншого боку, з використанням CABI розмір блоку малий. То ж коли при балансуванні навантаження виникає помилка, кількість часу, що витрачається на повільному вузлі для обробки цього блоку, менша порівняно з TCP. Таким чином, час реакції балансування навантаження є меншим. Результати цього експерименту показують, що при CABI час реакції балансування навантаження спадає з коефіцієнтом 8 в порівнянні з TCP.

Розглянемо *вплив спланованої схеми керування попитом на гетерогенні кластери.* Для цього експерименту розглянуто вплив на продуктивність керування запитом (МКП) буферного планування в БРД при використанні TCP і CABI. З тієї ж причини, що і при використанні схеми циклічного планування (згадується в попередньому пункті) розмір блоку 2 Кбайти був обраний для CABI і розмір блоку 16 Кбайт – для TCP. Рисунок 11 демонструє час виконання додатку. Передбачається, що вузол повинен ставати в часі динамічно повільнішим. Імовірна сповільненість вузла змінюється на осі абсцис. Таким чином, ймовірність 30% означає, що 30% обробки даних здійснюється в більш повільному темпі, а інші 70% здійснюються в темпі початкової фази вузла. Позначення CABI(*n*) на рисунку позначає додатки, що працюють з використанням CABI і коефіцієнтом неоднорідності *n*. Інші позначення легенди інтерпретуються таким же способом.



а

б

Рис. 9. Вплив високопродуктивних сокетів на середній час зворотної реакції запитів: *а* – без врахування затрат обробки; *б* – з лінійними затратами на обробку

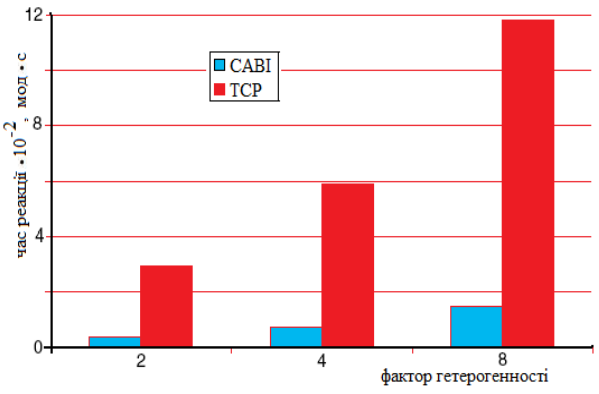


Рис. 10. Вплив неоднорідності на швидкість обробки при балансуванні навантаження з використанням циклічної схеми планування

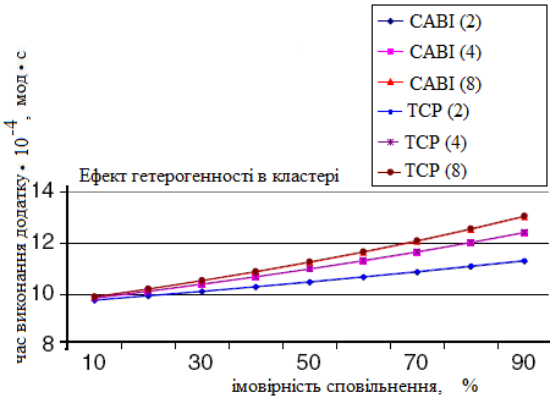


Рис. 11. Вплив неоднорідності на швидкість обробки при балансуванні навантаження за допомогою схеми планування керованого попиту

Видно, що продуктивність додатків, що використовують протоколи TCP є близькою до таких же, що використовують CABI. Це, в основному, стається через факт, що призначення блоків даних за допомогою керування запитом до споживачів дозволяє спрямовано виконувати більшу роботу на менш завантажених процесорах. До того ж, конвеєризація результатів даних забезпечує хороше переміщення їх між комунікацією і обробкою. Отже результати показують, що якщо високопродуктивні сокети не наявні на апаратній конфігурації, додатки повинні бути структуровані так, щоб скористатися конвеєризацією в процесі обробки і динамічним плануванням даних. Однак, як показують результати, використання високопродуктивних сокетів є бажаним для гарантій часу очікування і продуктивності.

5. Висновки та перспективи подальших досліджень

Разом з вимогами продуктивності додатки з інтенсивною обробкою даних мають також і інші вимоги, такі як гарантія продуктивності, масштабованість цих гарантій і пристосовність до гетерогенних мереж. Як правило, такі додатки, написані з використанням інтерфейсу сокетів на базі *kernel* та TCP/IP. Щоб дозволити таким додаткам мати перевагу використовувати високоефективні протоколи, дослідники наблизились до цього через опрацювання ряду методів, в тому числі і високопродуктивних шарів сокетів через протоколи на рівні користувача, таких як ABI і AHB. Тим не менше, ці рівні сокетів принципово обмежені тим, що додатки, які використовують їх, були написані зі збереженням продуктивності зв'язку в TCP/IP.

У роботі вивчаються можливості та обмеження для високопродуктивних сокетів CABI у ході роботи по відношенню до компонентного підходу, покликаного забезпечити текучу підтримку процесу виконання додатків з інтенсивною обробкою даних, що називається також БРД. Експериментальні результати показують, що шляхом реорганізації окремих компонентів додатків, досягаються значні покращення в продуктивності, що приводить до підвищення масштабованості додатків з гарантіями виконання і дрібноблокового балансування навантаження, яке, в свою чергу, робить їх більш адаптованими до гетерогенних мереж. Експериментальні результати також показують, що різні характеристики роботи CABI дозволяють ефективніший поділ даних на вихідних вузлах, що покращує продуктивність в деяких випадках аж до рівня підвищення на порядок. Це свідчить, що в поєднанні з високою продуктивністю сокети з низькими накладними затратами надають можливість додаткам досягти якісних показників в ході їх реалізації у багатьох напрямках вимірювання. Ці результати мають чітке застосовуються у проектуванні, розробці та реалізації додатків з інтенсивною обробкою даних нового покоління на сучасних кластерах.

Література

1. Guerin, R. Network Quality of Service, The Grid: Blueprint for a New Computing Infrastructure [Text] / R. Guerin, H. Schulzrinne; I. Foster, C. Kesselman, (Eds.). – Morgan Kaufmann, San Francisco, 1999. – 479–503.

2. Beynon, M. D. Distributed processing of very large datasets with DataCutter [Text] / M. D. Beynon, T. Kurc, U. Catalyurek, C. Chang, A. Sussman, J. Saltz // Parallel Computing. – 2001 – Vol. 27, Issue 11. – P. 1457–1478. doi: 10.1016/s0167-8191(01)00099-0

3. Oldfield, R. Armada: A parallel file system for computational [Text] / R. Oldfield, D. Kotz // In Proceedings of CCGrid2001, 2001. – P. 194–201. doi: 10.1109/ccgrid.2001.923193

4. Plale, B. dQUOB: Managing Large Data Flows by Dynamic Embedded Queries [Text] / B. Plale, K. Schwan // IEEE High Performance Distributed Computing (HPDC), 2000. – P. 1–8. doi: 10.1109/hpdc.2000.868658

5. GigaNet Corporations [Electronic resource] / Available at: http://www.giganet.com

6. Boden, N. J. AGigabitper-Second Local Area Network. [Electronic resource] / N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, W. K. Su Myrinet. – Available at: http://www.myricom.com

7. Frazier, H. Gigabit Ethernet: from 100 to 1000Mbps [Text] / H. Frazier, H. Johnson // IEEE Internet Computing. – 1999. – Vol 3, Issue 1. – P. 24–31. doi: 10.1109/4236.747317

8. Infiniband Trade Association [Electronic resource] / Режим доступу: http://www.infinibandta.org

9. Petrini, F. The Quadrics Network (QsNet): High-Performance Clustering Technology [Text] / F. Petrini, W. Feng, A. Hoisie, S. Coll, E. Frachtenberg // HOT 9 Interconnects. Symposium on High Performance Interconnects. – 2001. – Vol. 9. – P. 125–130. doi: 10.1109/his.2001.946704

10. GigaNet Corporations [Text] / cLAN for Linux: Software Users' Guide.

11. Myricom Corporations [Text] / The GM Message Passing System.

12. Pakin, S. High Performance Messaging on Workstations: Illinois Fast Messages (FM) [Text] / S. Pakin, M. Lauria, A. Chien // In Proceedings of Supercomputing, 1995. – P. 55. doi: 10.1145/224170.224360

13. Buonadonna, P. BVIA: An Implementation and Analysis of Virtual Interface Architecture [Text] / P. Buonadonna, A. Geweke, D. E. Culler // In Proceedings of Supercomputing, 1998. – P. 7–13. doi: 10.1109/sc.1998.10052

14. Kim, J. S. SOVIA: A User-level Sockets Layer over Virtual Interface Architecture [Text] / J. S. Kim, K. Kim, S. I. Jung // In the Proceedings of IEEE International Con-ference on Cluster Computing, 2001. – P. 1–10. doi: 10.1109/clustr.2001.960006

15. Shah, H. V. High Performance Sockets and RPC over Virtual Interface (VI) Architecture [Text] / H. V. Shah, C. Pu, R. S. Madukkarumukumana // In the Proceedings of

CANPC workshop (held in conjunction with HPCA Conference), 1999. – P. 91–107. doi: 10.1007/10704826_7

16. Afework, A. Digital dynamic telepathology - the Virtual Microscope [Text] / A. Afework, M.D. Beynon, F. Bustamante, A. Demarzo, R. Ferreira, R. Miller, M. Silberman, J. Saltz, A. Sussman, H. Tsang // In Proceedings of the 1998 AMIA Annual Fall Symposium. American Medical Informatics Association, November, 1998. – P. 1–10.

17. Catalyurek, U. The virtual microscope [Text] / U. Catalyurek, M. D. Beynon, C. Chang, T. Kurc, A. Sussman, J. Saltz // IEEE Transactions on Information Technology in Biomedicine. To appear. – 2003. – Vol. 7, Issue 4. – P. 230–248. doi: 10.1109/titb.2004.823952

18. Beynon, M. Design of a framework for data-intensive wide-area applications [Text] / M. Beynon, T. Kurc, A. Sussman, J. Saltz // In Proceedings of the 9th Heterogneous Computing Workshop (HCW2000), IEEE Computer Society Press, 2000. – P. 116–130. doi: 10.1109/hcw.2000.843737

19. Balaji, P. Impact of High Performance Sockets on Data Intensive Applications [Text] / P. Balaji, J. Wu, T. Kurc, U. Catalyurek, D. K. Panda, J. Saltz // Technical Report OSU-CISRC-1/03-TR05, The Ohio State University, Columbus, OH, 2003. – P. 1–10. doi: 10.1109/hpdc.2003.1210013

References

1. Guerin, Roch and Schulzrinne, Henning (1999). Network quality of service, in The Grid: Blueprint for a Future Computing Infrastructure, I. Foster and C. Kesselman, eds., Morgan Kaufmann Publishers, 479–503.

2. Beynon, M. D., Kurc, T., Catalyurek, U., Chang, C., Sussman, A., Saltz, J. (2001). Distributed processing of very large datasets with DataCutter. Parallel Computing, 27 (11), 1457–1478. doi: 10.1016/s0167-8191(01)00099-0

3. Oldfield, R., Kotz, D. (2001). Armada: A parallel file system for computational. In Proceedings of CCGrid2001, 194–201. doi: 10.1109/ccgrid.2001.923193

4. Plale, B., Schwan, K. (2000). dQUOB: Managing Large Data Flows by Dynamic Embedded Queries. IEEE High Performance Distributed Computing (HPDC), 1–8. doi: 10.1109/hpdc.2000.868658

5. GigaNet Corporations. Available at: http://www.giganet.com

6. Boden, N. J., Cohen, D., Felderman, R. E., Kulawik, A. E., Seitz, C. L., Seizovic, J. N., Su, W. K. Myrinet: AGigabitper-Second Local Area Network. Available at: http://www.myricom.com

7. Frazier, H., Johnson, H. (1999). Gigabit Ethernet: from 100 to 1000Mbps. IEEE Internet Computing, 3 (1), 24–31. doi: 10.1109/4236.747317

8. Infiniband Trade Association. Available at: http://www.infinibandta.org

9. Petrini, F., Feng, W., Hoisie, A., Coll, S., Frachtenberg, E. (2001). The Quadrics Network (Qsnet): High-Performance Clustering Technology. HOT 9 Interconnects. Symposium on High Performance Interconnects, 9, 125–130. doi: 10.1109/his.2001.946704

10. GigaNet Corporations. cLAN for Linux: Software Users' Guide.

11. Myricom Corporations. The GM Message Passing System.

12. Pakin, S., Lauria, M., Chien, A. (1995). High Performance Messaging on Workstations: Illinois Fast Messages (FM). In Proceedings of Supercomputing, 55. doi: 10.1145/224170.224360

13. Buonadonna, P., Geweke, A., Culler, D. E. (1998). BVIA: An Implementation and Analysis of Virtual Interface Architecture. In Proceedings of Supercomputing, 7–13. doi: 10.1109/sc.1998.10052

14. Kim, J. S., Kim, K., Jung, S. I. (2001). SOVIA: A User-level Sockets Layer over Virtual Interface Architecture. In the Proceedings of IEEE International Conference on Cluster Computing, 1–10. doi: 10.1109/clustr.2001.960006

15. Shah, H. V., Pu, C., Madukkarumukumana, R. S. (1999). High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In the Proceedings of CANPC workshop (held in conjunction with HPCA Conference), 91–107. doi: 10.1007/10704826_7

16. Afework, A., Beynon, M. D., Bustamante, F., Demarzo, A., Ferreira, R., Miller, R., Silberman, M., Saltz, J., Sussman, A., Tsang, H. (1998). Digital dynamic telepathology - the Virtual Microscope. In Proceedings of the 1998 AMIA Annual Fall Symposium. American Medical Informatics Association, 1–10.

17. Catalyurek, U., Beynon, M. D., Chang, C., Kurc, T., Sussman, A., Saltz, J. (2003). The virtual microscope. IEEE Transactions on Information Technology in Biomedicine, 7 (4), 230–248. doi: 10.1109/titb.2004.823952

18. Beynon, M., Kurc, T., Sussman, A., Saltz, J. (2000). Design of a framework for data-intensive wide-area applications. In Proceedings of the 9th Heterogneous Computing Workshop (HCW2000), IEEE Computer Society Press, 116–130. doi: 10.1109/hcw.2000.843737

19. Balaji, P., Wu, J., Kurc, T., Catalyurek, U., Panda, D. K., Saltz, J. (2003). Impact of High Performance Sockets on Data Intensive Applications. Technical Report OSU-CISRC-1/03-TR05, The Ohio State University, Columbus, OH, January, 1–10. doi: 10.1109/hpdc.2003.1210013

Рекомендовано до публікації д-р техн. наук, професор Максимович Л. В.
Дата надходження рукопису 20.05.2015

Мельник Василь Михайлович, кандидат фізико-математичних наук, доцент, кафедра комп'ютерної інженерії, Луцький національний технічний університет, вул. Львівська, 75, м. Луцьк, Україна, 43018
E-mail: melnyk_v_m@yahoo.com
Мельник Катерина Вікторівна, кандидат технічних наук, доцент, кафедра комп'ютерної інженерії, Луцький національний технічний університет, вул. Львівська, 75, м. Луцьк, Україна, 43018
E-mail: ekaterinamelnik@gmail.com
Багнюк Наталія Володимирівна, кандидат технічних наук, доцент, кафедра комп'ютерної інженерії, Луцький національний технічний університет, вул. Львівська, 75, м. Луцьк, Україна, 43018