DOI: 10.36910/6775-2524-0560-2020-40-20
УДК: 004.451
Мельник Василь Михайлович[1], к.ф.-м.н., доцент,
http://orcid.org/0000-0001-8282-6639
Каганюк Олексій Казимирович[1], к.т.н., доцент,
https://orcid.org/0000-0003-4616-8768
Козленко Микола Іванович[2], к.т.н., доцент,
https://orcid.org/0000-0002-2502-2447
Чернящук Наталія Леонідівна[1], д.п.н., професор,
http://orcid.org/0000-0002-3178-8377
Щерблюк Андрій Миколайович[1], магістрант
[1]Луцький національний технічний університет
[2]Прикарпатський університет ім. В.Стефаника

ЗАЛЕЖНІСТЬ ІНТЕНСИВНОСТІ ОБРОБКИ ДАНИХ В КЛАСТЕРІ ВІД ПРОДУКТИВНОСТІ СОКЕТІВ БЕЗ ВРАХУВАННЯ ГЕТЕРОГЕННОСТІ.

В.М. Мельник, О.К. Каганюк, М.І. Козленко, Н.Л. Чернящук, А.М. Щерблюк. Залежність інтенсивності обробки даних в кластері від продуктивності сокетів без врахування гетерогенності. В роботі розкриваються особливості функціонування та обмеження сокетів за архітектурою віртуального інтерфейсу, які використовують компонентну основу, створену для підтримки додатків з інтенсивною обробкою даних в момент їх виконання. Шляхом дифрагментації даних можна досягати значних покращень в продуктивності роботи таких додатків, що призводить до підвищення їх масштабованості та гарантії виконання, дозволяє здійснювати балансування розподілу навантаження та робить їх більш пристосованими до неоднорідних ресурсів. Результати досліджень доводять, що різні робочі характеристики сокетів за архітектурою віртуального інтерфейсу дозволяють ефективно розподіляти дані у вузлах, покращуючи на порядок продуктивність роботи додатків.
Ключові слова: сокети за архітектурою віртуального інтерфейсу, продуктивність мережі, інтенсивна обробка даних, віртуальний інтерфейс, віртуальні кластери

В.М. Мельник, А.К. Каганюк, М.І. Козленко, Н.Л. Чернящук, А.Н.Щерблюк. Зависимость интенсивности обработки данных в кластере от производительности сокетов без учета гетерогенности. В работе раскрываются особенности функционирования и ограничения сокетов по архитектуре виртуального интерфейса, которые используют компонентную базу, созданную для поддержки приложений с интенсивной обработкой данных в момент их выполнения. Путем дифрагментации данных можно достигать значительных улучшений в производительности работы таких приложений, что приводит к повышению их масштабируемости и гарантии выполнения, позволяет осуществлять балансировку распределения нагрузки и делает их более приспособленными к неоднородным ресурсам. Результаты исследований показывают, что различные рабочие характеристики сокетов по архитектуре виртуального интерфейса позволяют эффективно распределять данные в узлах, улучшая на порядок производительность работы приложений.
Ключевые слова: сокеты по архитектуре виртуального интерфейса, производительность сети, интенсивная обработка данных, виртуальный интерфейс, виртуальные кластеры

V. Melnyk, A. Kahaniuk., M. Kozlenko, N. Cherniashchuk, A. Shcherbliuk. Data processing intensity dependence in the cluster on socket performance avoiding heterogeneity. The work reveals the features of the sockets functioning and limitations alighted on the virtual interface architecture, which use the component base created to support applications with intensive data processing at the time of their execution. Significant improvements in such applications performance can be achieved by message diffraction, which leads to increase in their scalability and execution guarantee, allows balancing the load distribution and makes them more adapted to heterogeneous resources. The research results show that the various performance characteristics of sockets alighted on the virtual interface architecture allow the data efficient distribution in nodes improving the performance of worked applications.
Keywords: virtual interface architecture sockets, network performance, intensive data processing, virtual interface, virtual clusters.

Вступ.

В наш час багатогранні типи досліджень, які використовують високопродуктивну обробку даних, все більше наголошують на аналітичні методи та підходи, їх втілення під час розробки мережевих обчислювальних додатків. Додатки для виконання швидкісних обчислень в кластері даних застосовуються у таких галузях, як науовій, технічній, медичній, космонавтиці та інших [1,2]. Сьогоденні сенсорні технології дозволяють відтворювати досить високу роздільну здатність багатовимірних даних в мікроскопічних зйомках. В такому разі виникає великий інтерес до створення пакетних додатків для інтенсивної обробки даних, які б в діалоговому режимі могли виконувати фундаментальні дослідження, узагальнювати та аналізувати великі обсяги даних [3] у різних наукових галузях.

Комп'ютерні кластери, які використовують у своїй архітектурі стандартні апаратні засоби, все більше стають провідною базою суперкомп'ютерів, призначених для різнотипних додатків, які виконують інтенсивну обробку великих масивів даних і вимагають також потужної ресурсної підтримки збоку робочих та системних платформ. Така вимога ставиться в можливості ефективного переміщення великих обсягів

даних між пам'яттю об'єднаного процесора. З метою отримання високої продуктивності роботи такого кластера всі операції, які пов'язані з обміном даних та їх обробкою, також мають бути чітко керованими за допомогою засобів динаміки управління всередині кластера. З іншої сторони, такі додатки вимагають постійної підтримки високого рівня гарантій продуктивності, можливість пристосуватися до неоднорідності робочих середовищ і наявності змін в наданих системних ресурсах.

Робота мережевих додатків тісно пов'язана з базовими структурами, які, в основному, мають компонентну будову [4-6] і призначені для підтримки роботи середовища, забезпечуючи його гнучкість та продуктивність для швидкісних додатків на розподілених платформах. Беручи основу з базових структур, мережевий додаток представляє собою набір програмних взаємодіючих компонент, які розміщені разом з обчислювальними ресурсами. Таке розміщення відіграє важливу роль в прояві гнучкості роботи додатків та оптимізації їх продуктивності. Також в наш час паралельна обробка даних зазвичай виконується за допомогою мультикопіювання в околі кластера зберігання компонента та вузлів для обробки [4].

Застосування конвейєризації є ще одним зручним методом реалізації для зростання продуктивності такої системи. У додатках з високошвидкісною обробкою даних застосовується і підхід перерозподілу повного об'єму даних на субблоки, визначені користувчем наперед, обробка яких може здійснюватись конвеєрно, тобто ефективніше, враховуючи їх новоутворений розмір (розмір дифрагментації). У разі, коли обробка даних і їх доставка, тобто зв'язок між машинним процесором, можуть в ході організації роботи перекриватися, то зростання продуктивності такої системи також буде залежати від обчислювальної блоковості та об'єму повідомлень даних, тобто субблоків надсилання. Відомо [7], що невеликі за об'ємом субблоки надісланих даних ведуть до зростання балансу навантаження і покращення процесу конвеєризації. Завдяки цьому велика кількість повідомлень після процесу дифрагментації генеруються блоками малих розмірів, менших за характерний розмір повідомлення корисного навантаження. Проте, з іншої сторони, передавання даних у вигляді блоків більшого об'єму веде до зменшення кількості повідомлень надсилання. Таким чином, зростає пропускна здатність каналу передачі, проте має місце прояв дизбалансу навантаження в системі, що призводить до зниження конвеєрності передачі та обробки пакетів даних.

Аналіз останніх досліджень

Завдяки застсоуванню сучасних високошвидкісних комунікаційних систем з міжкомпонентною архітектурою [4,8,9] "критична ділянка" в організації зв'язку та обміні повідомленнями для програмних додатків значно змістилася. В роботах [10,11] велися активні дослідження цієї ділянки з метою створення на рівні користувача нових потоколів для систем передачі повідомлень, які б призводили до зменшення часу затримки та зростання пропускної здатності в передачі даних. Однак, деякі дослідницькі компанії активізували власні роботи зі стандартизації високопродуктивних протоколів на рівні користувача, до яких сьогодні можна віднести *архітектуру віртуального інтерфейсу*, дані досліджень про яку, наприклад, відображені в роботі [12].

В роботах [13,14] високошвидкісні додатки та їх робота були орієнтовані на базові протоколи TCP чи UDP, які використовують традиційний інтерфейс сокетів. З метою підтримки додатків і застосування їх без будь-яких змін було реалізовано специфічні підходи, які містили в собі сокетні з'єднання на користувацькому рівні з урахуванням роботи високопродуктивних протоколів. Таким чином програми, написані з урахуванням у використанні базових протоколів ядра Linux, брали звичайний TCP/IP продуктивний зв'язок за основу. Слід додати, що на відміну від сокетних зв'язків на основі ядра Linux високошвидкісні засоби володіють ще і відмінними характеристиками, що має вплив на критичну ділянку продуктивності, яка забезпечується ними. Зміна деяких впливових компонентів для діючого мережевого додатка, наприклад, розміра блоку дифрагментованих даних, утворених з загальних вихідних даних, дозволяє отримати перевагу в робочих характеристиках для підвищення продуктивності таких засобів, збільшуючи їх швидкодію обробки та адаптованість в загальній системі кластера. Дані обмеження та елементи реалізації також стосуються високошвидкісного механізму сокетів за архітектурою віртуального інтерфейсу саме на проміжках, де покращення обміну повідомленнчми та їх параметри ведуть до підвищення продуктивності роботи і гнучкості компонентної системи. Тобто механізм дифрагментації великого блоку надісланих даних на субблоки характерного для системи об'єму в реалізації її компонентного підходу покликаний забезпечити адаптацію в роботі та вчасну підтримку прийому-передачі даних [4] для додатків з інтенсивною обробкою, інтегрованих в ній.

Як уже відомо [15], в наш час високопродуктивний сокет може бути цілком втілений в спеціальний додаток із вкладеною в нього необхідною інтерактивністю з метою досягнення гарантій продуктивної роботи системи по відношенню до кінцевого споживача через покращення адаптованості програм для інтенсивної обробки в середовищах з гетерогенними умоваии. Результати літературних даних показують, що за допомогою спеціальної реорганізації компонентної системи для високошвидкісної обробки даних є можливість досягати вагомих підвищень продуктивності додатків обробки, яка, в свою чергу, буде обумовлювати гарантії виконання високошвидкісних обчислень та стимулювати високу масштабованість.

Значну роль відіграє у цьому введення в дію підходу роздрібнення даних та структурування навантажувального балансу, який також веде до збільшення адаптованості додатків обробки до гетерогенних середовищ і в процесі роботи знижує ресурсозмінність для них.

Зростання потужності обчислювальної техніки та дискової пам'яті привело до збільшення потенціальних можливостей створення та використання мережевих додатків та збереження блоків даних в терабайтних об'ємах. Для розуміння та *аналізу збереження великомасштабних даних* застосовуються коди візуалізації. Вводиться інтерактивна візуалізація, яка дає можливість заволодіти уявленнями конкретної складної системи з внутрішньої сторони, а аналіз даних та візуальне представлення об'ємних їх блоків інколи відіграють навіть і вирішальну роль у багатьох галузевих наукових дослідженнях. В наш час це в великій мірі стосується і додатків, працюючих у інтерактивному високошвидкіснлму режимі здійснення запитів та аналізу великих за об'ємом блоків даних, особливо в напрямках природничих наук. Широкодіючим додатком з високопродуктивною обробкою є аналіз отриманих даних з мікроскопу з метою відтвлоення їх з дуже великою роздільною здатністю та оцифровуванням [16]. Подібний додаток в своїй роботі використовує такі необхідні характеристики, як екземпляр проведеного (знятого) прикладного дослідження і мотивуючий сценарій обробки отриманих даних.

Швидкісні обчислювальні додатки звжди повинні працювати в пакетному режимі роботи і мають змогу генерувати та обробляти великі за обсягами масиви даних, обробка яких залежить від зовнішніх та внутрішніх параметрів швидкодії. Такі параметральні залежності уже досліджувалися в роботі [16] на кластері віртуальних машин для високопродуктивної обробки, який проявляв бінарну сумісність для сокетних додатків зі стандартним інтерфейсом. "За допомогою введеного в систему механізму спрощеної комунікації, реалізованого на базі Xen 3.2 та ядра Linux виявлено покращення внутрішніх та зовнішніх параметрів мережевої швидкодії, який демонструє взаємодію віртуальних машин, подібно до організації зв'язку на основі UNIX DOMAIN сокетів. Встановлено, що пропускна здатність між комунікуючими машинами з використанням спрощеного зв'язку зростає приблизно на 2,11 %, ніж для традиційних TCP/IP-протоколів, а швидкодія передачі повідомлень – на 7,8–7,9 %" [16].

Досить вагомою задачею є підтримка в роботі програмного забезпечення, необхідна для пошуку, опрацювання і збереження оцифрованих слайдів досліджень з урахуванням забезпечення процесу інтерактивності під час так званих робочих інтервалів фізичного мікроскопу у стандартних проявах його поведінки [17,18]. Основна трудність, що проявляється в процесах обробок даних цифрових зображень з високою роздільною здатністю, пов'язана з величиною їх обсягів, які можуть варіативно змінюватися в межах від сотень мегабайт до гігабайт чи декількох їх сотень. Для цього базове системне та програмне забезпечення з великою точністю повинно чітко відображати поточні дані робочого мікроскопу, враховуючи безперервне маніпулювання його збільшення та послідовне кадрове слідування. Обробка клієнтських запитів потребує високороздільного проектування даних на виділену область резолюції, враховуючи відповідний піксельний розподіл для однієї точки розділу. Відповідно, сервер для візуалізації цифрової мікроскопії повинен приймати і зберігати різноманітні типові запити, які можуть направлятися збоку клієнта. Найпоширенішими є запити повного оновлення, які під час їх виконання повинні представити абсолютно нове зображення. Також можуть мати мвсце і запити на типово часткові оновлення, які подають зображення в положенні дещо зміщеного або збільшеного варіану представлення. Такий сервер призначається для виконання запитів обох таких типів.

Цифрова обробка даних за допомогою додатків, від яких подаються запити і які маніпулюють набором даних наукового характеру досить часто може представлятися ациклічним грубим потоком у вигляді наперед визначених блоків даних від одного чи декількох джерел до клієнтських процесорних вузлів. Для прикладу, це може бути один чи декілька наборів даних, які витримують відповідний розподіл у відповідності до систем їх збереження. Дані, які першими представляють інтерес для такого запиту, забираються з визначених наборів і після цього проходять обробку за допомогою відповідної послідовності операцій, які виконуються на вузлах обробки. Таким чином, дані в додатку цифрової мікроскопії, які викликають науковий інтерес, проходять обробку за допомогою додавання цифрових субблоків та відповідного перегляду [4,19] і тільки після цього вони надсилаються на сторону клієнта.

Збережені дані у вигляді субблоків утворюють частинні зображення чи фрагменти індексованих даних, які необхідні для інтегрування з метою отримання блоку. Повне зображення може складається з визначеної множини блоків. Під час виконання запиту часткового оновлення для його отримання може знадобитися лише конкретна частина блоків. Отже, з метою виконання таких запитів байтовий розмір і протяжність блоку суттєво впливають на об'ємність даних, які необхідні для операцій отримання та передавання між вузлами.

Підтримка роботи додатків з інтенсивною обробкою даних

Можна розглядати різні напрямки впливу з метою покращення продуктивності високошвидкісної системи обробки та додатків, які працюють в цій системі. Першим напрямком є застосування

редифрагментації наборів даних у системі з метою удосконалення паралелізму під час здійснення операцій вводу-виводу та отримання даних, які прямо стосуються виконання запиту. Завдяки добре розрахованій блоковій відповідності, підбору їх декластеризації, запит виконання може потрапити до максимальної кількості задіяних дисків. Другим є ефективне використання потужності обробки даних збоку системи для випадку, коли робочий додаток для обробки даних розроблений для застосування паралелізму. І, як уже згадувалося, третім напрямком покращення продуктивності в інтерактивному режимі роботи додатків для обробки дифрагментованих наборів даних є їх конвеєризація обробки. Тобто загальний витрачений час обробки додатком може бути в різній мірі зменшеним за умови розділення даних на субблоки чи порції і виконання обробки шляхом конвеєризації. Для більшості додатків конвеєрна обробка забезпечує завершальний механізм, який покроково утворює кінцевий варіант в результаті інтеграції даних після обробки. Ткаим чиком, уже не потрібно очікувати на завершення запиту, а вести відслідковування результатів частково оброблених блоків. На даному етапі це не зменшує сукупний час обробки запиту, однак є досить ефективним в роботі інтерактивних додатків при умові, що під час дослідження постійно зміщується кругозір зацікавленості.

Як вище було сказано, на конвеєрне виконання обробки даних та продуктивність високошвидкісної системи великий вплив має *розподіл субблоків* та їх *розмір*. Параметр розміру субблоків повинен враховувати пропускну здатність мережі в процесі обміну та час затримки. Другий параметр враховує передавання повідомлення з повною обробкою за допомогою протоколів використання системою на повному треку його проходження від відправника і аж до кінцевих одержувачів. Як говорилося вище, кількість повідомлень, необхідна для передачі однакового об'єму даних, спадає зі збільшенням розміру субблоку дифрагментації. В такому випадку пропускна здатність як параметр продуктивності системи відіграє важливішу роль, ніж час затримки. Проте, у разі збільшення розміру субблоку час обробки на кожен з них теж зростає, система втрачає чутливість у зв'язку зі зменшенням кількості окремих чи послідовних її оновлень. Однак, якщо розмір субблоку зменшується, то реально збільшується кількість необхідних субблоків для пердавання одного і того ж об'єму даних по мережі. В цьому випадку час очікування повідомлення може зайняти роль домінуючого фактора у сукупній ефективній роботі додатку. З іншої сторони, субблоки невеликих розмірів сприяють покращенню збалансованості навантаження на додаток між копіями компонентів (субблоками передавання), проте продуктивність може знижуватися через затрати на з'єднання.

Передавання даних субблоками більших розмірів приведе до покращення часу реакції системи для здійснення повного оновлення запиту, який пропорційний пропускній здатності системи. У випадку виконання часткового запиту на оновлення буде спостерігатися збільшення кількості даних та затрат на їх обробку. Зі зменшенням розміру субблоку запит на часткове оновлення не буде отримувати багато лишніх даних, як це буває під час виконання їх, проте запит на повне оновлення буде зазнавати втрат, обумовлених спадом пропускної здатності.

Збільшуючи величину діючої пропускної здатності зі зниженням часу затримки повідомлень, високопродуктивні сокети володіють і іншими можливостями [15], такими як зростання мережевої пропускної здатності у разі зменшення розміру повідомлення. Такий прояв можна зафіксувати, порівнюючи їх роботу з базовими *kernel*-сокетами, наприклад, такими як TCP/IP. В даній роботі відмічається, що зниження величини часу затримки при проходженні повідомлення сталого об'єму в байтах досить відчутне. Суттєво замітним є також зниження часу затримки, що в загальному і призводить до зростання продуктивності роботи системи.

Неоднорідність навантаження в умовах гетерогенності може аргументуватися в наступних ситуаціях. По-перше, апаратне середовище може складатися з машин різної робочої потужності і ємності пам'яті. По-друге, ресурси можуть розподілюватися між іншими додатками. В результаті, наявність ресурсів, таких як CPU і пам'ять може динамічно змінюватися. У таких випадках додаток повинен бути структурно спроектованим і пристосованим до гетерогенної природи середовища для 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 є планування заздалегідь вихідних даних і розрахунків додатка 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 швидкодіючі вузли отримували 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 ресурсів та зміну доступності до них.

Під час розробки додатків і підтримки їх роботи компонентно-базові засоби [5,6] 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 зазнати вдосконалення шляхом конвеєрного виконання багатьох копій цих компонентів. Таким чином, в даній роботі використовується базова компонентна інфраструктура блокового розбиття даних [4], призначена для підтримки додатків з інтенсивною обробкою даних в розподілених середовищах. Також реалізується високоефективний інтерфейс сокетів за архітектурою віртуального інтерфейсу, розроблений для додатків, які використовують TCP/IP, щоб надати переваги в реалізації можливостей продуктивності.

Методика дифрагментації даних на блоки

Далі коротко опишемо структуру засобу ділення даних на блоки [4], 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) стартового потоку. Інтерфейс для фільтра складається з трьох функцій: функції ініціалізації, в якій розміщені та інііалізуються всі необхідні ресурси, в тому числі і такі як пам'ять для структури даних; функції обробки, в яких операції, визначені користувачем, застосовуються на елементах даних; і функції завершення, в яких проходить звільнення ресурсів, що задіяні в функціях ініціалізації.

Фільтри з'єднюються за допомогою логічних потоків – однонапрямлених потоків даних від одного фільтра (відправника розсилань) до іншого (отримувача). Фільтр виконує функції читання даних з його вхідних потоків і запису данихі тільки 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-з'єднання для потокової взаємодії між цими точками.

Загальна структура додатка обробки реалізується за допомогою групи фільтрів, з'єднаних між собою за допомогою логічних потоків. Коли група фільтрів запускається для обробки запиту додатка, робоча система в цей момент здійснює з'єднання сокетів між фільтрами, розміщених на *різних хостах*, перш ніж почати виконання запиту програми. Фільтри, розміщені на одному комп'ютері, діють як окремі нитки. Запит додатка обробляється як єдиний робчий модуль відповідною групою фільтрів. Прикладом може бути візуалізація набору даних в залежності від кута огляду. Обробка даних єдиним робчим модулем може бути здійснена в конвеєрному режимі, в якому різні фільтри можуть працювати з різними елементами даних одночасно. Обробка єдиним робчим модулем 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, функція роботи з даними, що надходять і розміщуються по буферах, і функція запису даних в будь-які вихідні потоки. Після обробки останнього буфера передається спеціальний маркер від системи обробки, щоб відзначити кінець для поточного єдиного робчого модуля (рис 1,a). Функція завершення або фіналізації викликається після завершення обробки поточного єдиного робчого модуля з метою звільнення задіяних ресурсів, таких як робочий простір. З метою обробки наступного єдиного робчого модуля можуть активуватися в подальшому функції інтерфейсу.

Програмна модель надає кілька розподілень для полегшення оптимізації продуктивності. Прозора копія фільтра – це копія фільтра у групі фільтрів, схематично показана на рисунку 1,b. Вона прозора в тому розумінні, що вона 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 повиннен бути незмінним, незалежно від числа прозорих копій. Прозорі копії також 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. Логічний потік відповідає за елементи планування (або буфери) в потоці даних між прозорими копіями фільтра. Для прикладу, 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 гост3 або гост4. Для розподілу між прозорими примірниками система виконання підтримує циклічно мігруючий механізм і механізм керування попитом на основі рівня буфера споживання. Механізм керування попитом спрямований на відправку буферів на фільтр, що найшвидше мав би обробляти їх. Коли фільтр споживача починає обробку буфера, отриманого від фільтра-відправника, він посилає повідомлення підтвердження фільтру продюсору з метою відмітити, що буфер знаходиться в процесі обробки. Фільтр-відправник вибирає фільтр використання з мінімальною кількістю непідтверджених буферів для відправки даних буферу обробки, досягаючи таким чином кращого балансу між навантаженням.

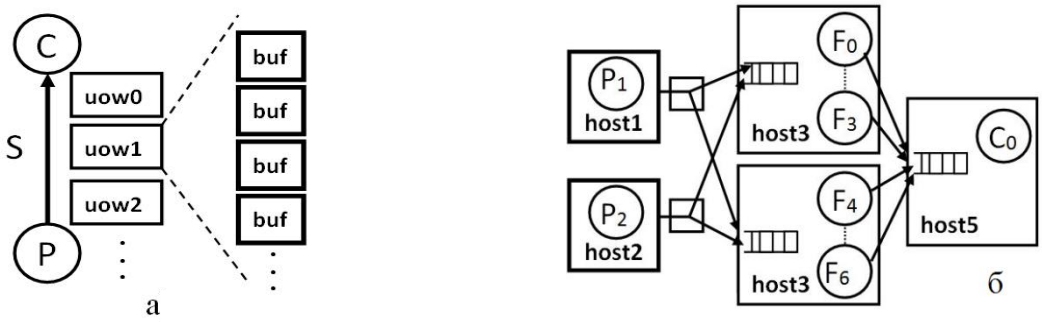

Рис. 1 – Блокове розбиття даних в потоковому абстрагуванні і підтримка копій:
a) розміщення буферів даних і маркерів кінця роботи потоку;
b) встановлення груп фільтрів P, F, C при використанні прозорих копій

Мета і завдання дослідження

Метою даної роботи ставилося дослідження залежності інтенсивності обробки даних в кластері від продуктивності сокетів з урахуванням його гетерогенності та вторинного розбиття на субблоки. При цьому повинна використовуватися базова компонентна інфраструктура блокового розбиття вихідних даних, яка призначена в розподілених середовищах для підтримки високопродуктивних додатків, та високоефективний інтерфейс сокетів за архітектурою віртуального інтерфейсу, розроблений для TCP-додатків, з метою надання переваги в реалізації продуктивної їх роботи. Для виконання поставленої мети необхідно виконати наступні завдання:

- на базі персональних комп'ютерів реалізувати кластерну систему з залученням традиційних сокетів TCP та сокетів за архітектурою віртуального інтерфейсу з механізмом перерозбиття даних та можливістю дослідження неоднорідності обробки;
- оцінити максимальну продуктивність обробки даних, використовуючи сокети за архітектурою віртуального інтерфейсу та TCP-сокети у вигляді міні-тестів на час затримки і пропускну здатність з урахуванням прямого і непрямого впливів на роботу додатків та блокового розбиття даних;
- дослідити параметр середнього часу затримки на обробку блока даних з гарантією на оновлення за кожну часову секунду;

Теоретичні основи сокетів за архітектурою віртуального інтерфейсу

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 традиційних протоколів, таких як TCP і UDP. Мережа cLAN є апаратним втіленням архітектури віртуального інтерфейсу (в літературі – Virtual Interface Architecture). Є два типових варіанти сокетної реалізації у мережі cLAN. Один 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 LAN-емулятора на рівні сокетів – це така реалізація, яка використовує рівні від IP до віртуального інтерфейсу [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, заснований на використанні бібліотек рівня користувача, які базуються на примітивах архітектури віртуального інтерфейсу рівня користувача. Реалізація в даній роботі потрапляє в цю категорію. Спочатку виконується посилання на визначений сокетний рівень, такий як сокети за архітектурою віртуального інтерфейсу, описаний в решті частини викладеного матеріалу. Так як реалізація сокетів за архітектурою віртуального інтерфейсу не є основною в даній статті, то результати мікро-тестів для нашого рівня сокетів будуть представлені в наступному. Для інших деталей, пов'язаних з розробкою та застосуванням сокетів за архітектурою віртуального інтерфейсу, можна звернутися до роботи [20].

Далі, для виконання *оцінки ефективності* буде представлено дві групи результатів. По-перше, буде досліджуватись максимальна продуктивність, надана через сокети за архітектурою віртуального інтерфейсу у вигляді міні-тестів на час затримки і пропускну здатність. По-друге, буде розглянуто прямий і непрямий вплив на продуктивність, представлену високопродуктивними сокетами в додатках, реалізованих з

використанням блокового розбиття даних для того, щоб оцінити обидві характеристики: час затримки і аспекти пропускної здатності на шляху контролю.

Експерименти проводилися на кластері персональних комп'ютерів, який складається з (16 виходів) 420 вузлів, з'єднаних мережами Giganet cLAN і Fast Ethernet. Використовуються гост-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-бітових 33 МГц PCI роз'єми. Ці вузли оснащені 512 Мбайт SDRAM і 256 К кеш-пам'яттю L2 рівня, версія ядра Linux-2.2.17.

Опис експериментальної установки

Для дослідження продуктивності були використані два типи додатків. Перший додаток (тип) був призначений для емулювання візуалізаційного сервера, тобто за допомогою даної програми реалізлваний чотирьохступінчатий конвеєрний механізм, в якому на останньому етапі включений фільтр візуалізації. В кожному дослідженні з метою поліпшення прикінцевої пропускнної здатності з кожного фільтра виконувались три екземпляри копіювання, які знімалися в зазначеному конвеєрі. Розміщення фільтрів зображені на рисунку 2. Зображення візуалізувалося користувачем у наміченому вузлі візуалізації за допомогою включеного фільтра, після чого зі сховища зберігання вибиралися необхідні дані і передавалися для подальшої роботи на інші наступні фільтри. Кожен наступний фільтр в системі схематично розміщений на іншому вузлі, включеному у зазначений конвеєр. Будь-яке зображення, що буде розглядатися, вимагатиме дані в кількості 16 Мбайт для отримання і виконання їх обробки. Як можна зрозуміти, такі намічені дані наперед визначеного розміру будуть містити фрагменти зображення, а визначений розмір цієї пам'яті називають блоком розподілу. Однак для такого розміру блоку розподілу повне зображення буде складатися з кількох відповідних субблоків. Якщо користувач подає на виконання запит про часткове або повне оновлення зображення, то системою мають бути доставлені необхідні блоки, які повинні бути включені у відтворення даного сукупного зображення. Вимагається, щоб кожен блок доставлявся цілісний, що може вимагати ініціалізацію додатково затратних даних і навть їх мережеве передавання до місця сукупного відтворення.

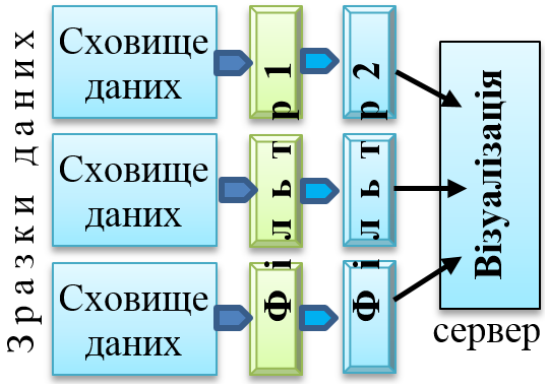

Рис. 2 – Архітектура експериментальної установки

В роботі були взяті до уваги два типи запитів, перший з яких вимагає повного оновлення зображення або первинний запит на візуалізацію нового зображення, – тобто запит вимагає повного об'єднання всіх необхідних блоків в єдине зображення, які повинні брати участь у виконанні цього запиту. В процесі виконання даний тип оновлення є чутливим до пропускної здатності і з практичної точки зору спонукає до використання блоків великих розмірів. В результаті, для виконання повного оновлення запитів з урахуванням певної швидкодії відтворення повинен би бути визначений розмір такого блоку, зберігання і передача якого б задовільняли згадані умови користувача.

Реалізація другого запиту полягає в частковому оновленні зображення, який зазвичай виконується у випадку часткового зміщення користувачем вікна візуалізації, чи намірено змінює мастаб зображення, яке візуалізується в даний момент. Для продуктивного виконання цього типу запиту потрібно лише до діючих додати додаткові блоки доповненого відображення, тобто у невеликій їх кількості в порівнянні з кількістю утворюючих повне зображення блоків. Продуктивне виконання такого запиту є залежним в основному від часу затримки в ході їх передавання. В даному випадку фрагменти відображення зазвичай витягуються чи звужуються в ході відновлення, а, отже, було б краще задіювати в цю процедуру невеликі за об'ємом блоки даних.

У випадках, коли розмір блоку є досить великим за об'ємом, часткове оновлення зображення займе більший час експозиції, тому що блок має тенденцію видозмінюватися, навіть і у випадку, коли у загальному відображенні задіяна його невелика частка. З використанням занадто малих розмірів блоків

виконання запитів на повне оновлення займатиме значно довший час, так як має бути видозмінено досить немало менших додаткових блоків за розміром. Одже, для додатку, який спроможний обробляти два згадані типи запитів, необхідно встановлювати спеціальне узгодження на їх виконання. В наступних проведених експериментах виявлено покращення продуктивності додатку та його масштабованості з використанням сокетів за архітектурою віртуального інтерфейсу та гарантій виконання запитів для кожного типу оновлення, порівнюючи з TCP сокетами, результати яких будуть нижче викладені. Інший підхід реалізації в додатку включає в себе механізм балансування розподілу навантаження на програмне забезпечення через використання засобу розбиття даних на блоки. Якщо виконується обробка даних кількома робочими вузлами, то їх покращена конвеєризація буде досягнута тоді, коли потрібний для механізму балансування навантаження час є достатнім для надсилання одного блоку даних обробки до вузла і дорівнює затратному часу обробки блоку даних на даному вузлі. Типово розрахований розмір блоку для такого додатка вибирається з метою досягнення необхідної збалансованості в процесі конвеєризації блоків з урахуванням часових інтервалів обробки і комунікації. В експерименті береться до уваги те, що потужність обчислювальних вузлів є незмінною на протязі всього процесу виконання додатка.

Результати та їх обговорення

Представимо дані дослідження параметра середнього часу затримки блока даних з гарантією на оновлення за кожну секунду часу. Результати першої групи експериментальних даних стосуються серії тих досліджень, в яких користувач старається удосконалити наперед заданий рівень покадрової обробки. Він залежить від кількості заново згенерованих зображень чи повних оновлень, які були виконані за часову секунду. В околі дії такого обмеження проводиться дослідження середнього значення часу затримки, який отримується під час виконання запиту на часткове оновлення зображення. На рисунках 3 і 4 представлені залежності ефективності роботи додатка від величини оновлення за часову секуду. В залежності від заданої кадрової частоти для запиту з новими зображеннями TCP-сокет ставить певну вимогу про встановлення відповідного розміру блоку даних для досягнення наміченої пропускної здатності.

Залежність часу затримки для виконання запиту часткового оновлення з фрагментованими даними на частини відповідно до вказаної вимоги для TCP-протоколів мала би відповідати значенню часу затримки для даного фрагменту повідомлення. Залежність для того ж розміру фрагментації даних і використання сокетів за архітектурою віртуального інтерфейсу забезпечує більш високий рівень продуктивності системи. Однак з метою досягнення відповідно високої пропускної здатності для запитів повних оновлень дані сокети ставлять вимогу про суттєве зменшення розміру субблоку даних. Так, під час перерозподілу даних на субблоки з урахуванням часу затримки, обернено пропорційному пропускній здатності для сокетів за архітектурою віртуального інтерфейсу, даний параметр може бути помітно зменшений. На рисунку 3 наведено експериментальні залежності продуктивності (усередненого часу затримки) без будь-якої можливої обробки даних для всіх трьох згаданих ситуацій. Експериментальні результати підтверджують очевидну перевагу в продуктивності, яка досягається у випадку використання сокетів за архітектурою віртуального інтерфейсу без присутності будь-яких обчислювальних затрат на вузлах обробки даних. Як видно із наведеної залежності для випадку використання сокетів TCP не може бути досягнутим більше обмеження на оновлення, як 3,25 повних оновлень за часову секунду. Проте сокети за архітектурою віртуального інтерфейсу з перерозподілом даних на субблоки без помітного зниження продуктивності роботи здатні досягти вказаного кадрового рівня. Дані досліджень, отримані в експериментах без змін в перерозподілі субблоків, демонструють поліпшення продуктивності більше ніж в 3,5 рази, а для випадку з наявним блоковим перерозподілуом даних (нижня залежність) – більше ніж в 10 разів. Можна зробити висновок, що в умовах використання сокетів за архітектурою віртуального інтерфейсу та перебудовою деяких компонентів (наприклад, розмір субблоку) для додатка спостерігається суттєве покращення продуктивності його роботи, яке дає можливість досягати йому очевидних переваг як в роботі, так і в масштабованості з дотриманням продуктивних гарантій.

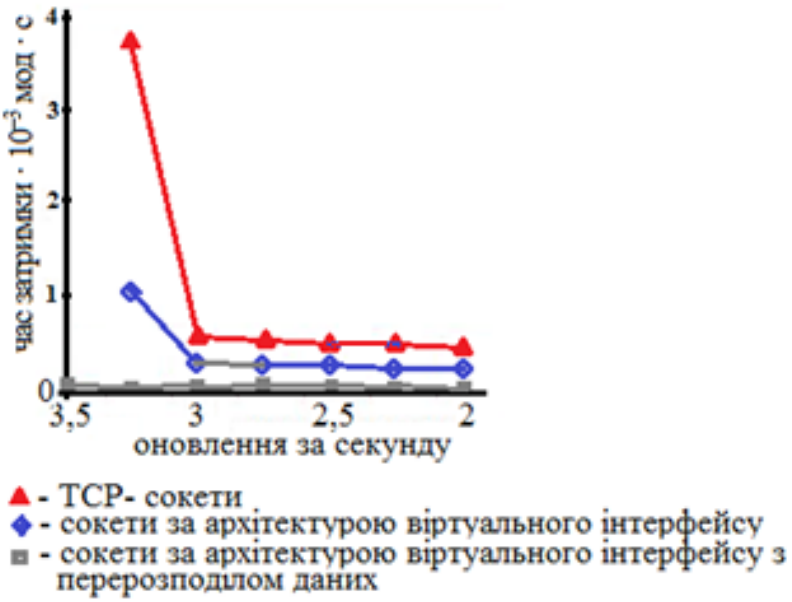

Рис. 3 – Вплив продуктивності сокетів на середній час затримки з гарантіями оновлень за секунду без обчислення затрат

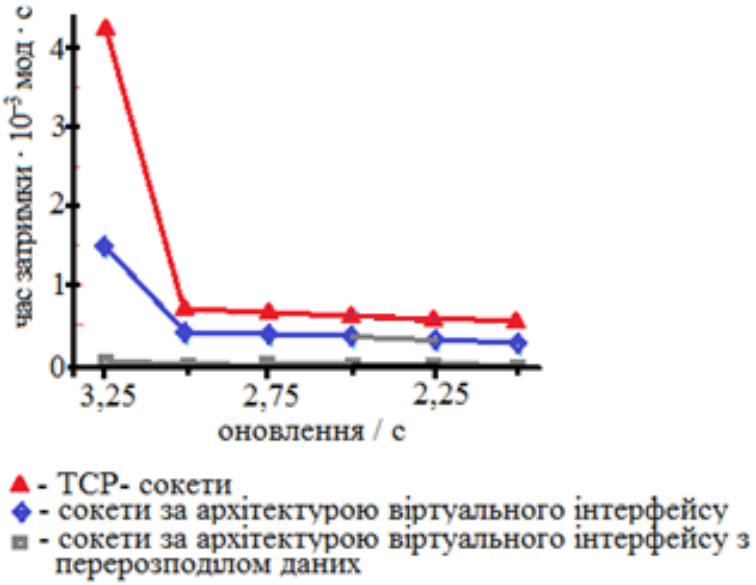

Рис. 4 – Вплив продуктивності сокетів на середній час затримки з гарантіями оновлень за секунду з проведеним лінійним обчисленням затрат

Наведені на рисунку 4 експериментальні залежності демонструють продуктивність роботи додатка з присутніми затратами обробки даних на вузлах, яка лінійно пропорційно залежна від розміру субблоку обробки. Здійснювалося вимірювання величини часу обробки, який потрібний для візуалізації частини зображення додатка цифрової мікроскопії [17] з залученням процедури блокового розбиття даних. Встановлено, що дана часова величина становить ~18 пс на один байт даних. Подібні додатки, призначені для перегляду цифрованих слайдів після мікроскопічних з'йомок, володіють низькими втратами на піксель в ході обробки. В основному, сюди належать програми з перевагою в зниженні часу затримки та зростанні пропускної здатності на проміжних вузлах. Ціллю дослідження також є потреба сконцентруватися на подібого типу додатках.

Порівнюючи з даними, отриманими з попереднього експерименту, в даному випадку навіть сокети за архітектурою віртуального інтерфейсу з поділом даних на субблоки не перевершили рівень 3,25 оновлень за секунду. Такі показники обумовлені тим, що пропускна здатність у випадку сокетів за архітектурою віртуального інтерфейсу обмежувалася затратами обробки на кожному із вузлів. Однак, в експерименті має місце загальне покращення продуктивності, зокрема більше ніж на 3,9-4,1 рази у випадку безблокової обробки даних, і приблизно на ~12 разів з наявним перерозподілом даних на субблоки.

Тепер розглянемо ефект *оновлення за секунду з гарантіями часу затримки*, який стосується другої частини пророблених експериментів. В цій ситуації спостерігається тенденція збільшення кількості повних оновлень за секунду під час діючого конкретного значення параметра часу затримки, яке є макимальним для часткового запиту на оновлення, що виконується. На наступних рисунках наведені результати підвищення продуктивності обробки даних, досягнутої додатком для заданого часу затримки. Для протоколів TCP спостерігається залежність продуктивності від блоку даних, розмір якого не може бути більшим, ніж деяке певне значення. Під час виконання розбиття даних з урахуванням згаданої вимоги поліпшення досягнутої пропускної здатності зазнає відповідних обмежень, що і відслідковується в ході аналізу залежності для TCP, поданої на рисунку 5. В процесі дослідження з сокетами за архітектурою віртуального інтерфейсу і використанням таких же за розміром блоків даних досягнуто значно кращої продуктивності, що і відслідковується з аналізу відповідної залежності для цих сокетів. Проте виконання рефрагментації даних з урахуванням параметрів пропускної здатності та часу затримки за їх значеннями, про що мова велася вище в обговоренні результатів, досягається значно вищий рівень продуктивності, який теж відображений на рисунку у вигляді найнижчої кривої залежності для даних сокетів з поділом даних. Слід також нагадати, що рисунок 5 розглядає всі наведені криві продуктивності без наявності будь-якої обробки даних на вузлах. На рисунку 6 представлені експериментальні криві залежностей продуктивності з урахуванням обробки на вузлах, яка, як уже згадувалося, також лінійно залежить від розміру блоків. При відсутності затрат на обробку та з обмеженням часу затримки в межах ~100 μs продуктивність обробки для сокетів TCP стрімко спадає, в той час як для сокетів за архітектурою віртуального інтерфейсу продуктивна робота продовжується зі значеннями характеристик, близьких до пікових. В цілому результати, отримані з експерименту для цього типу досліджень, демонструють значне поліпшення продуктивності, приблизно в 6 разів, без врахування будь-якого перерозподілу блоків повідомлень.

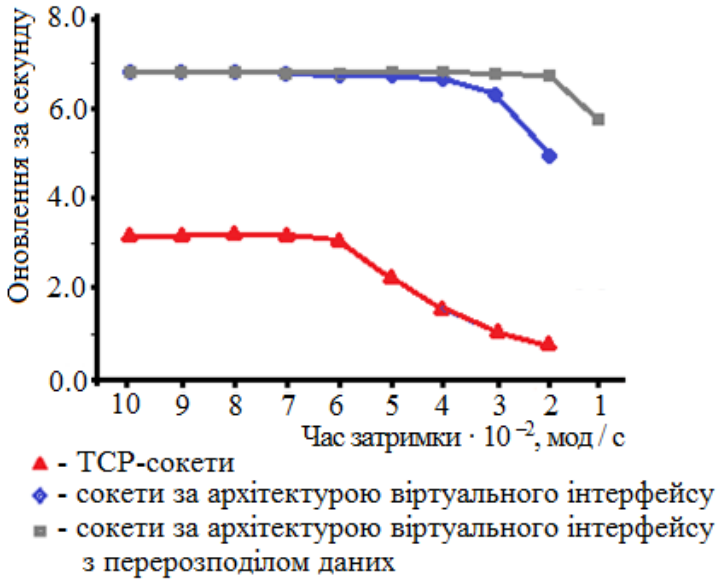

Рис. 5 – Вплив ефективності сокетів на оновлення за секунду без врахування затрат на обробку на вузлах з гарантіями часу затримки Час затримки · 10 –2, мод/с

Ще значніше зростання продуктивності, уже більше ніж у 8 разів, спостерігається під час використання дій блокового перерозподілу під час обробки для сокетів за архітектурою віртуального інтерфейсу (рис. 6). Якщо врахувати затрати обробки даних на вузлах, то можна помітити, що робота сокетів за архітектурою віртуального інтерфейсу і TCP та великою гарантією виконання для заданого часу затримки є досить подібна. Причина подібності випливає з урахування затрат на вузлову обробку та шляхів з'єднань. Наприклад, для сокетів за архітектурою віртуального внтерфейсу з затратами обробки 18 nc/байт, рівень обробки даних не проявляє зростання, а виходить все-таки на рівень сталої поведінки, в той час як у випадку використання сокетів TCP система передачі швидше досягає даного рівня. На відміну від сокетів TCP з тієї ж причини кадровий рівень, досягнутий сокетами за архітектурою віртуального інтерфейсу та спаданням часу затримки, змінюється незначно. Загалом результати наведеного експерименту доводять зростання продуктивності приблизно в 4 рази.

Рис. 6 – Вплив ефективності сокетів на оновлення за секунду з гарантіями часу затримки та лінійними затратами обробки на вузлах

Висновки та перспективи досліджень

В роботі реалізовано кластерну систему на базі персональних комп'ютерів з залученням традиційних TCP та сокетів за архітектурою віртуального інтерфейсу з механізмом перерозбиття даних та можливістю неоднорідності обробки. За результатами пророблених міні-тестів дано оцінку величині максимальної продуктивнлсті обробки даних у випадках використання сокетів за архітектурою віртуального інтерфейсу та TCP-сокетів на основі експериментально отриманих значень часу затримки і пропускної здатності з урахуванням прямого і непрямого впливів на роботу додатків таблокового розбиття даних. Для сокетів за архітектурою віртуального інтерфейсу без врахування затрат обробки на вузлах спостерігається поліпшення продуктивності більше ніж в 3,5 рази, а для цих же сокетів з наявним блоковим перерозподілуом даних – більше ніж в 10 разів. Ще значніше зростання продуктивності, уже більше ніж у 8 разів, спостерігається під час використання дій блокового перерозподілу з гарантіями часу затримки за часову секунду.

За рахунок компонентного підходу та блоковим розбиттям для забезпечення підтримки роботи додатків з інтенсивною обробкою даних отримані експериментальні результати доводять значні покращення в продуктивності за рахунок реорганізації окремих компонентів додатків, що стимулює зростання їх масштабованості з гарантіями виконання.

Дані пророблених експериментів доводять також, що параметральні характеристики роботи сокетів за архітектурою віртуального інтерфейсу та використання поділу даних на вихідних вузлах обробки веде до покращення продуктивності, а в деяких випадках аж на порядок. В ході реалізації сокети з високою продуктивністю та низькими затратами на обробку даних дозволяють у багатьох напрямках додаткам досягти значно вищих якісних показників. Отримані результати мають чітке застосовуються на сучасних кластерах та у проектуванні і реалізації високопродуктивних додатків з інтенсивною обробкою даних.

References.

1. S. Iannucci, V. Cardellini, O. D. Barba, I. Banicescu. (2020) A hybrid model-free approach for the near-optimal intrusion response control of non-stationary systems. // Future Generation Computer Systems. – 2020, V 109, p. 111-124.
2. D. Black, P. Jones (2015) Differentiated Services and Real-Time Communication, Informational. RFC 7657, – Nov. 2015. – IETF, ISSN 2070-1721.
3. Guerin R. and Schulzrinne H. The Grid: Blueprint for a New Computing Infrastructure, chapter Network Quality of Service. Morgan Kaufmann, 1999.
4. Beynon M. D., Kurc T., Catalyurek U., Chang C., Sussman A., Saltz J. Distributed processing of very large datasets with DataCutter. *Parallel Computing*, 27(11):1457–1478, October 2001.
5. R. Oldfield, D. Kotz. Armada: A parallel file system for computational. In Proceedings of CCGrid2001, May 2001.
6. B. Plale, K. Schwan. dQUOB:Managing large data flows using dynamic embedded queries. In HPDC, August 2000.
7. V. Melnyk, K. Melnyk, S. Lavrenchuk, I. Burchak, O. Kaganiuk. Influence of the message direct search mechanism based on the TCP protocols to the exchange process. East-European journal of Enterprise Technologies. – Kharkov (Scopus DOI:10.15587/1729-4061.2019.167995). – 2019. – №3(2)99. – p. 36-42.
8. 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. http://www.myricom.com.
9. Petrini F., Feng W.C., Hoisie A., Coll S., Frachtenberg E. The Quadrics Network (QsNet): High-Performance Clustering Technology. In Hot Interconnects, 2001.
10. Myricom Corporations. The GM Message Passing System.

11. S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In Proceedings of Supercomputing, 1995.
12. P. Buonadonna, A. Geweke, and D. E. Culler. BVIA: An Implementation and Analysis of Virtual Interface Architecture. In *Proceedings of Supercomputing*, 1998.
13. Kim J.S., Kim K., Jung S.I. SOVIA: A User-level Sockets Layer over Virtual Interface Architecture. In the Proceedings of IEEE International Conference on Cluster Computing, 2001.
14. Shah H.V., Pu C., Madukkarumukumana R.S. High Performance Sockets and RPC over Virtual Interface (VI) Architecture. In the Proceedings of CANPC workshop (held in conjunction with HPCA Conference), pages 91-107, 1999.
15. Melnyk V., Bahniuk N., Melnyk K. High-performance sockets influence on data computing intensity. // Scientific Journal ″ScienceRise″ (DOI: 10.15587/2313-8416.2015.44380). – Kharkiv. – 2015. – Vol 6, № 2(11). – P. 38-48.
16. Melnyk V., Melnyk K., Kuz'mych O., Bahniuk N. Kravez' O. Improvement investigation of internal and external parameters for communication speed in a cluster of communicated virtual machines. // Scientific journal "Computer-integrated technologies: education, science, production". – Lutsk. – 2020, №39. – p. 162-174.
17. Afework A., Beynon M.D., Bustamante F., Demarzo A., Ferreira R., Miller R., Silberman M., Saltz J., Sussman A., Tsang H. Digital dynamic telepathology - the Virtual Microscope. In Proceedings of the 1998 AMIA Annual Fall Symposium. American Medical Informatics Association, November 1998.
18. Catalyurek U., Beynon M.D., Chang C., Kurc T., Sussman A., Saltz J. The virtual microscope. IEEE Transactions on Information Technology in Biomedicine. To appear.
19. Beynon M., Kurc T., Sussman A., Saltz J. Design of a framework for data-intensive wide-area applications. In Proceedings of the 9th Heterogeneous Computing Workshop (HCW2000), pages 116–130. IEEE Computer Society Press, May 2000.
20. Balaji P., Wu J., Kurc T., Catalyurek U., Panda D.K., Saltz J. Impact of High Performance Sockets on Data Intensive Applications. Technical Report OSU-CISRC-1/03-TR05, the Ohio State University, Columbus, OH, January 2003.