УДК 004.451
DOI: 10.15587/1729-4061.2019.167995

Вплив механізму прямого пошуку повідомлень на базі TCP-протоколів на процес їх обміну

В. М. Мельник, К. В. Мельник, С. В. Лавренчук, І. Н. Бурчак, О. К. Каганюк

Реалізовано механізм прямого пошуку з розширенням традиційного сокетного TCP-інтерфейсу для отримання повідомлень, обходячи традиційний порядок встановленої черги. Даний механізм може застосовуватись для високопродуктивних та кластерних комп'ютерних систем з метою інтенсифікації обміну даними та неперервності підтримки максимального навантаження на обчислювальні машини. Інтерфейс для прямого пошуку повідомлень реалізований на базі ядра операційної системи Linux. Отримання експериментальних результатів тестування здійснено за допомогою простого набору microbenchmark-міток. В ході тестування відправник надсилає необхідне число повідомлень сталого розміру на встановлене з'єднання, а приймач пропускає неочікувані повідомлення і зчитує очікувані повідомлення в простір користувача. Спосіб відшукання очікуваних повідомлень реалізований завдяки багаторазовому пошуку для випадку, коли сокетний додаток розглядає TCP-сокет як список повідомлень з можливістю приймати і видаляти дані не тільки з вершини, але з будь-якого місця в сокетному буфері. Всі очікувані повідомлення розпізнаються і обробляються розробленим викликом seek_recv(). Кожен тест містить ~80 повторень, які включають операції відкриття сокета, пересилання 800–1000 повідомлень відповідно до політики прийому і закриття сокета. Система використовує тільки один активний сокет в один і той же час.

Отримані результати доводять помітне зниження процесорного часу обробки повідомлень на 36–40 % та загальне зростання продуктивності. Однак, при наближенні до об'єму повідомлень 1000 байт, близького до характерного розміру корисного завантаження TCP-пакета, спостерігається падіння продуктивності процесу обміну

Ключові слова: TCP-сокети, пошук повідомлення, розширення інтерфейсу, продуктивність

1. Вступ

Високопродуктивні кластерні обчислення широко використовуються для виконання довготривалих громіздких розрахунків. Відомо, що такі розрахунки можна розділити на окремі обчислювальні частини, кожну з яких можна виконувати на окремих комп'ютерах. Для виконання подібних задач комп'ютери повинні надсилати дані один одному для здійснення розрахунків з відповідною частотою їх надходження. Такий підхід також застосовують і з метою отримання об'єднаної цілісної інформації, пов'язаної з розподілом обчислень на різних комп'ютерах, а також в організації подання чергової порції даних для здійснен-

ня наступних обчислювальних кроків. Слід добавити, що кластерні компоненти сьогодні можуть забезпечити ефективне середовище для додатків з інтенсивною обробкою даних на розподілених платформах [1–4]. Для цього дані повинні бути сформовані в спеціальні структури, а додатки в такому середовищі повинні розроблятися таким чином, щоб легко та вміло оперувати ними. Для роботи з подібними структурами даних додаток розробляється з набору програмних компонентів, розміщення яких разом з ресурсами обчислення відіграє ведучу роль для гнучкості та оптимізації продуктивності їх роботи.

В багатьох додатках з інтенсивною обробкою об'єм даних може дифрагментуватися на субблоки визначеного об'єму байт, вигідного для конвеєрної їх обробки. Слід зазначити, що на практиці організації подібних обчислень обробка і зв'язок можуть перекриватися з метою покращення продуктивності такої системи. В роботі [5] доводиться, що невеликі блоки даних в ході передавання призводять до покращення конвеєризації і балансу навантаження, що має місце в практиці комунікації. Зазначається також, що блоки даних більшого розміру, хоча і зменшують кількість повідомлень для передавання, однак стимулюють підвищення пропускної здатності системи та призводять до невідповідностей в навантаженні, що веде до зменшення конвейєрності обробки даних.

Додатки, написані з використанням сокетного інтерфейсу на базі TCP/IP, поряд із продуктивністю та інтенсивністю обробки даних, ставлять і інші вимоги, такі як пристосованість до роботи з гетерогенними мережами, а також гарантія продуктивності обробки та масштабованість. Для отримання переваги з використанням високоефективних протоколів в кластерних додатках було використано спеціальні розроблені підходи, в тому числі і з залученням високопродуктивних сокетних рівнів на рівні користувача через діючі протоколи. До них відносяться і архітектура віртуального інтерфейсу та AHB [5], додатки для яких повинні бути написані з урахуванням збереження продуктивності зв'язку. В ході реорганізації окремих програмних компонентів отримуються покращення продуктивності процесу прийому-передачі даних та результатів обчислень, спостерігається підвищення масштабованості додатків та адаптація до гетерогенних мереж. За архітектурою віртуального інтерфейсу для значного покращення продуктивності деякі характеристики роботи сокетів підтримують ефективніший поділ даних на вихідних вузлах. З високою продуктивністю та низькими накладними затратами сокети надають можливість додаткам досягти і якісних показників у багатьох напрямках.

Незважаючи на те, що кластерна структура ставить вимогу робочим комп'ютерам бути якомога повністю завантаженими з метою зменшення часу на обчислення, все ж кластерні системи зазнають накладних затрат в ході протікання комунікаційного процесу. Такі накладні затрати залежать також і від кількості комп'ютерів в кластері, бібліотек використання для організації зв'язку, від вибору інтерфейсу для внутрішньої комп'ютерної комунікації та комунікації між іншими блоками всередині обчислювального кластера. Незважаючи на різноманітні експериментальні результати, останні дослідження продемонстрували взаємозалежність ефективного дизайну бібліотеки для організації обміну повідомленнями в кластерах, які використовують компоненти

Ethernet та TCP/IP для досягнення продуктивності з'єднань [6, 7]. То ж ставиться за мету удосконалення комунікаційного процесу зосереджене на прийомі та передачі повідомлень на основі TCP-протоколів, обумовлене деякою перебудовою конкретних сокетних рівнів.

2. Аналіз літературних даних та постановка проблеми

Для вирішення проблеми покращення ефективності комунікаційного процесу різні автори зосереджувались на двоякому підході. Один із них враховував перебудову дизайну бібліотеки, відповідальної за обмін повідомленнями на базі протоколів TCP, а інший – перебудову збоку апаратного забезпечення. Так, в роботі [8] запропоновано реалізацію підходу підтримки збоку апаратного забезпечення, використовуючи інтерфейс мережевої плати для відключення критичних частин під час обробки протоколів.

Однак велика кількість авторів зосереджуються на підтримці передачі повідомлень за допомогою специфічних сокетних з'єднань через використання перебудованих інтерфейсів на рівні користувача чи на рівнях індивідуальних протоколів [9–13]. Такі підходи також приводять до підвищення швидкодії, продуктивності, зменшення часу затримки та покращення загального комунікаційного процесу. Однак вони використовують затратні операції перекопіювання даних, не враховують комунікацію, спрямовану на конкретний мережевий додаток користувача, дотримуючись порядку встановленої черги на сокеті.

В роботі [14] пропонується протокол потокового управління передачею (ППУП), який є надійним протоколом транспортного рівня, аналогічний до протоколу TCP у тому відношенні, що володіє механізмом загально-потокового керування з'єднаннями між двома діючими робочими станціями. Однак на відміну від TCP, таке з'єднання можна назвати асоціативним, що може охоплювати декілька незалежних потоків повідомлень в єдиний комунікаційний процес. ППУП забезпечує впорядкованість повідомлень в межах потоку, але не по всій потоковій асоціації. Такий механізм міг би бути використаний для забезпечення переваг для сокетів прямого пошуку в залежності від конкретних обставин, за наявності яких дані можуть бути оброблені поза встановленим порядком. Наприклад, тегові MPI-запити на отримання даних можуть бути реалізовані як окремі асоційовані потоки, тобто незалежні веб-відповіді на *pipe*-лінійному з'єднанні. Однак прямий пошук наміченого повідомлення в черзі та його отримання додатком користувача поза встановленим порядком черги ще не реалізувалися в даних роботах.

Однак сокети прямого пошуку є більш гнучкими, тому що вони дозволяють з відхиленням від встановленого порядку отримання відповіді на запит поза чергою вибірки повідомлень. Крім того, API-інтерфейс користувача для сокетів пошуку є лише єдиним розширенням на стороні прийому по відношенню до інтерфейсу традиційних сокетів. В свою чергу ППУП вимагає набагато складніших розширень та модифікацій для функцій надсилання, отримання та функцій управління з'єднаннями [15]. Даний засіб комунікації є перпендикулярним (ортогональним) до сокетів прямого пошуку. Тут дозволяється викорис-

товувати по декілька IP-адрес для кожного комп'ютера, що входить до встановленої потокової асоціації, потенційно використовуючи кілька мережевих шляхів для збільшення процесу надійної передачі.

Як уже згадувалося вище, залежність витрат, що виникають під час обміну повідомлень на базі TCP-протоколів, походить не тільки від їх розміру та частоти передавання, але й від того, чи повідомлення є "очікуваним" чи "неочікуваним" (несподіваним) для мережевого додатка. В роботі [16] зазначається, що повідомлення може надходити в чергу як неочікуване у випадку, якщо його дані надходять до процесу-приймача перед здійсненням виклику бібліотеки з метою одержати таке повідомлення в буфер пам'яті на рівні програми користувача. Відомо [7, 17], що неочікувані дані копіюються спочатку в тимчасовий буфер бібліотеки. Такі операції перекопіювання є затратними для загального процесу обміну і знижують його продуктивність.

В ході здійснення передачі на базі TCP-протоколів повідомлення можна вважати отриманим, якщо дані про нього з'явилися в мережі, а TCP-стек розміщує його в буфер сокета з'єднання між двома комунікаційними хостами. В роботі [16] представляється ситуація, коли програма користувача очікує на конкретне повідомлення поряд з неочікуваними повідомленнями в черзі, які надійшли до системи в той же момент чи швидше. Наприклад, система може очікувати на повідомлення з інтерфейсом його передавання та заданим типом тега [18]. Інтерфейс передавання повідомлень (ІПП) представляє собою стандартизовану і портативну систему передачі для реалізації на різноманітних паралельних обчислювальних архітектурах.

До подібних робіт можна долучити і ті, які базуються на використанні перебудованої бібліотеки з більш ефективним дизайном, керовані подіями збоку власної доповненої архітектури [19]. Сюди слід додати і роботи, що використовують спеціальну підтримку збоку апаратної загальноприйнятої перебудови та інтерфейсу мережевої карти, а також такі, які базуються на розщепленні рівня TCP [7, 8].

Очікуване підвищення продуктивності отримується за рахунок специфічної перебудови інтерфейсу сокетного рівня операційної системи. Виходячи з описаної моделі [16] та практичної реалізації, вона є зовсім не споріднена з роботами, що зосереджуються на підтримці з'єднання через використання інтерфейсу мережевої карти або архітектури бібліотеки використання. Дана робота повинна співнапрямлено працювати з ідеями, які ведуть до перебудов та вдосконалення інших системних компонентів з метою підвищення високопродуктивної TCP-комунікації.

На практиці кластерної комунікації існує декілька досить ефективних реалізацій інтерфейсу передачі повідомлень, багато з яких є вільнодоступними або відкритими для використання. Це дало поштовх до розпаралелення_розвитку програмного забезпечення та розробки великомасштабних і портативних додатків, які призначені до виконання паралельних та розподілених обчислень. В ході комунікаційного процесу сокетний інтерфейс операційної системи для TCP-протоколів отримує певні байти від здійсненого з'єднання, використовуючи системні виклики recv() або read(). Згідно порядку черги та зі сторони почер-

гового отримання повідомлень спочатку потрібно звільнити сокет від попередніх неочікуваних повідомлень щоб в наступному отримати доступ до очікуваного повідомлення. Кожне з неочікуваних повідомлень повинно бути скопійоване в сторону пулу прийому неочікуваних повідомлень, використовуючи додаткові операції на копіювання, які спричиняють суттєві накладні затрати збоку системи доставки. В такому разі слід додатково перед отриманням повідомлення на рівні додатка перевірити пул прийому неочікуваних повідомлень, і тільки після цього викликати функцію прийому очікуваного повідомлення на рівні сокета з'єднання.

З аналізу літературних даних видно, що підхід прямого пошуку повідомлень в стеку сокета не є достатньо описаним та дослідженим. Однак він є спорідненим для робіт, спрямованих на підвищення продуктивності обміну повідомленнями на базі протоколів TCP. Розширений сокетний TCP-інтерфейс для здійснення прямого доступу до випадково розміщених повідомлень в межах одного з'єднання ще не достатньо вивчений. Не з'ясовано також і вплив запропонованого механізму на процес обміну та його характеристики з урахуванням параметрів повідомлень.

3. Мета і завдання дослідження

Метою роботи є реалізація механізму прямого пошуку повідомлень та його вплив на продуктивність процесу обміну повідомленнями. Даний механізм у поєднанні з розширеним сокетним інтерфейсом прямого пошуку можуть бути застосовані у забезпеченні конвейєрності доставки даних для їх обробки в швидкісних комп'ютерних системах та забезпечення стабільності навантаження на обчислювальних машинах.

Для досягнення цієї мети були поставлені наступні завдання:

– практично реалізувати змодельований [16] механізм прямого пошуку повідомлень на базі встановленої TCP-комунікації між робочими машинами та вмонтованим тестовим методом простого мікробенчмаркінгу;

– застосувати розширений сокетний TCP-інтерфейс та функцією пошуку для втілення підходу отримання очікуваного повідомлення з будь-якого стекового положення, обминаючи операції затратного копіювання;

– здійснювати відшукання хаотично розміщених очікуваних повідомлень в черзі, використовуючи багаторазовий пошук, та звільняти буфери пам'яті після їх отримання;

– застосовуючи мікробенчмарк-тестування системи з реалізованим механізмом виявити скорочення процесорного часу обробки очікуваних повідомлень та зростання продуктивності обміну в залежності від зміни їх об'єму та кількості.

4. Особливості Linux-взаємодії для реалізації механізму прямого пошуку повідомлень

Сокетний інтерфейс для здійснення операції прямого пошуку очікуваних повідомлень на сокеті може бути розроблений і реалізований на базі ядра операційної системи Linux. Перевірка функціонування інтерфейсу здійснюється за

допомогою використання способу тестового набору мікробенчмарк-міток, дані для якого отримуються поза запитом, реалізованим для даного сокета [20]. Результати якісних розрахунків та припущень [16] свідчать за скорочення процесорного часу обробки повідомлень з використанням підходу прямого пошуку очікуваного повідомлення в стеку. Зростання продуктивності такої системи буде залежати від отримання більших за об'ємом повідомлень очікування або від кількості повідомлень, які потрібно обійти в черзі отримання.

У відповідності до поданих даних [16] та Linux-теорії стек TCP під управлінням ядра Linux працює з окремими сокетними буферами, позначеними в літературі як sock_buff'*s* чи skb's. Оскільки передані пакети в процесі їх надходження приймаються мережевим пристроєм, то дані про них розміщаються в кільцевий буфер, а sock_buff призначається для самих даних, тобто корисної частини повідомлення. Буфер sock_buff зберігає метадані для кожного пакета, а стек операційної системи Linux для TCP/IP, взаємодіючи з буфером пам'яті sock_buff, обробляє дані пакета, відповідно. Для кожного конкретного з'єднання буфер sock_buff розміщується в загальній черзі сокетного буфера. Для спрощення прийому пакетів в правильному порядку та управління ними, кожному пакету присвоюється послідовний номер в черзі, згідно якого вказується сума байтів, надісланих для кожного пакета зокрема в ході з'єднання. Такий спосіб вирішення дозволяє в ході повторного запиту на рівні мережі відновлювати пакети на стороні прийому та вилучати отримані дані з сокетних буферів. Додаток користувача не знає, хоча повинен би знати впорядкованість даних в пакетах або почерговість їх надходження. Завдання, пов'язані виключно з додатком користувача, полягають у тому, яким чином елементи даних надсилаються з джерела та їх фіксовані довжини.

В роботі [16] подано блок-схему алгоритму роботи функцій, задіяних в прийомі повідомлення, що надійшло. Зокрема, функція tcp_recvmsg() призначена для копіювання даних повідомлення надходження з буферів sock_buff'*s* у черзі сокета, які асоціюються з фактичними даними, розміщеними в кільцевому буфері. Функція перевіряє перший skb у сокетному буфері, а потім копіює дані в буфер простору користувача. Якщо skb має більше даних, ніж зафіксовано в запиті, то функція tcp_recvmsg() залишає "нагадування" на черзі буфера сокета. Якщо користувач запитує більший об'єм даних, ніж той, що розміщений на першому skb, то він виділяється разом з відповідними даними на кільцевому буфері, а вказані кроки, описані вище, продовжуються з наступним skb що знаходиться в черзі. В кінцевому результаті функція tcp_recvmsg() повертає всі дані запиту із буфера сокета після процедури повного зчитування. Вона також вилучає всі буфери skb'и, які містили отримані дані в повному об'ємі, і оновлює номер послідовності першого байта для здійснення наступної операції зчитування на сокеті.

Також TCP використовує зазвичай номери послідовностей для відстеження треку зчитуваня з буфера сокетів і визначає порядок продовження процедури зчитування. На кожному кроці змінна copied_seq, яка фіксує порядок копіювання, отримує номер послідовності, за яким буде здійснюватися зчитування даних. Таким чином, дана змінна міститиме повний порядок того, що вже було

скопійовано, і що буде зчитуватися ще з черги прийому. То ж якщо послідовність номерів copied_seq була більша, ніж номер першої базової послідовності skb, а частина їх вже була зчитана з неї, то повна довжина запиту даних буде копіюватися, починаючи з номера послідовності, зазначеного в copied_seq. Отже, використовуючи номери послідовності будуть здійснюватися визначення даних на сокеті, які запит прийому повинен зчитувати.

5. Опис та реалізація додаткової процедури пошуку для сокетів

Основна мета застосування сокетів, які відшукують повідомлення в черзі, полягає в отриманні даних про них, які розміщуються на сокетному буфері прийому у довільному порядку. Коли дані копіюються з сокета, то відповідні skb'и зі скопійованими даними повинні бути вилучені з загального сокетного буфера, а поточний список skb'ів повинен бути також зменшений на їх вилучену кількість. Оскільки дані, які було зчитано і вилучено під відповідними номерами послідовності, більше не наявні і не доступні для будь-яких операцій, то наступні звернення на сокеті повинні заздалегідь «знати» і враховувати, що ці дані більше не існують в буфері.

Такий підхід реалізується через список з'єднань, що містить початкові та кінцеві номери послідовності для кожної уже вибраної "дірки" даних в сокетному буфері прийому. Створення дірки звільняє простір пам'яті у буфері сокета і дозволяє нормалізувати керування поведінкою TCP-потоку незалежно від місця в буфері, з якого були зчитані і видалені дані. Коли запит на отримання повідомлення починає процедуру копіювання даних в простір користувача, він пропускає на шляху будь-яку дірку, яка йому зустрічається, і продовжує нормально отримувати дані з порядкового номера, який слідує після дірки. В процесі отримування даних програмою користувача список дірок звичайно зростає, проходить їх зливання, а разом з тим і динамічне скорочення буфера, пов'язане з їх видаленням.

Розроблена в даній роботі реалізація вимагає створення нового потокового протоколу SOCK_SEEK_STREAM, який використовує той же самий стек, що і TCP та звичайні сокети типу SOCK_STREAM. Однак базові їхні функції повинні бути видозмінені таким чином, щоб можна було здійснювати прямий пошук очікуваних повідомлень за допомогою сокетів типу SOCK_SEEK_STREAM.

У випадку, якщо сокетний запит здійснюється на сокет, який не передбачає здійснювати процедури прямого пошуку повідомлення очікування, або якщо виклик не відшукує необхідний пакет, який очікується для отримання, то його шлях через стек TCP та функції майже ідентичні до коду, що використовується звичайним ядром Linux. Однак коли запит пошуку здійснюється на сокеті, вимагаючи прямого пошуку повідомлення очікування, то шлях через стек TCP залишається однаковим, але може змінюватися шлях коду через окремі сокетні функції. В основному зміни будуть стосуватися коду функції tcp_recvmsg(), які введені до неї. Всі додатково введені модифікації необхідні також і для керування списком дірок, порядковими номерами послідовності і skb'ами, які були вже зчитаними.

Ще одна вагома зміна повинна бути внесена до функції tcp_recvmsg(), яка стосується процедури отримання сокета, на якому здійснюється прямий пошук очікуваного повідомлення. Така зміна полягає в тому, що механізм попереднього завантаження черги TCP відключається [16]. Незважаючи на те, що механізм попереднього завантаження черги TCP дозволяє краще керувати ресурсами потоку в процесі обміну, цей механізм спричиняє невелике зниження продуктивності роботи. З іншого боку, в загальній процедурі обміну повідомленнями неможливо легко змінювати чергу завантаження, для того щоб активізувати процес здійснення подальшого пошуку. В зв'язку з усім вищесказаним, механізм попереднього завантаження повинен бути вимкнений саме в момент прийому на сокет, який здійснює прямий пошук повідомлення очікування.

Після того, як сокет типу SOCK_SEEK_STREAM був створений, то відомі функції recv() і recvmsg() можна використовувати в роботі як звичайні функції. Нова розроблена функція seek_recv() може бути реалізована у вигляді системного виклику, описаного нижче, якому необхідно надати наступні аргументи:

ssize_t seek_recv(int s, void *buf, size t len, int flags, size_t offset);

Як можна порівняти зі сторони застосування функцій для звичайних сокетів, аргументи для виклику seek_recv() будуть ідентичними, як і для функції recv(), однак зі зміною, доданою для вказівки передавання кількості байтів для переведення в потік. Таке зміщення завжди повинно вказувати на перший байт, який має бути отриманий через застосування системного виклику recv().

Так як виклик seek_recv() змінює структуру msghdr, а потім викликає загальну функцію sock_recvmsg(), то для здійснення пошуку для одержувачів також можна використовувати функцію стандартної бібліотеки recvmsg(). Змінна msg_seek була додана до структури msghdr для того, щоб вказати зміщення пошуку. Модифікуючи структуру msghdr, передану в функцію recvmsg() як параметр, можна легше здійснювати пошук пакета, який очікується для отримання, без залучення в роботу нової спеціальної функції.

Для того, щоб мати змогу здійснювати пошук попередніх повідомлень великих розмірів, потрібно збільшувати максимальний розмір буфера прийому, який контролюється через системну змінну net.core.rmem_max та параметр sysctl. Завдяки введенню для використання функції setsockopt() ця змінна дозволяє встановити максимальний розмір буфера на кожне прийняте повідомлення, який можна перевстановлювати. Отже, параметр sysctl слід встановити відповідно на достатню величину кількості байтів, а буфер прийому повинен бути збільшений у разі необхідності протягом всього інтервалу дії програми користувача.

У випадку, коли буфер прийому повністю заповнюється і виконуються звичайні TCP-операції, то виклик отримання пошуку повідомлення в процесі свого виконання повертає помилку. У випадку, коли отримуються нові пакети, а сокетний буфер уже повністю заповнений, то їх буде скинуто (не прийнято) і надіслано повторно відправником у відповідності до нормального управління потоку TCP [21]. В даному випадку з метою звільнити більше місця в ядрі додаток повинен чітко реагувати на помилку про переповнення сокетного буфера і видаляти деякі непотрібні дані, що могли залишитися в буфері.

6. Результати дослідження механізму прямого пошуку повідомлень на базі TCP-протоколів

Механізм прямого пошуку повідомлень згідно модельних припущень [16] був реалізований на хостах, між якими встановлений зв'язок, який дозволяв надсилати та отримувати повідомлення. Основним завданням в ньому було реалізувати сокетний інтерфейс, сформований з традиційного, шляхом перебудови звичайних сокетних функцій та викликів, який дозволяв би розпізнавати очікувані повідомлення у черзі в межах одного з'єднання, здійснювати їх доставку в простір користувача і, поряд з цим, виконувати видалення звільнених буферів пам'яті.

Інтерфейс для сокетів прямого пошуку повідомлень був реалізований на базі ядра Linux 2.6.13. Отримання експериментальних результатів було здійснено за допомогою тестування з отриманням простого набору microbenchmark-міток. Для отримання такого набору на перший погляд можна використати хости (мінімум дві робочі машини), призначені для відправки і отримання повідомлень через встановлене з'єднання з можливістю отримання простого набору мікроміток поза запитом.

Для здійснення тестування відправник кілька разів надсилає повідомлення сталого сконфігурованого розміру на встановлене з'єднання. Приймач кілька разів пропускає мимо в черзі N надісланих повідомлень, зчитує очікуване повідомлення, а потім зчитує N пропущених повідомлень, які знаходились перед призначеним очікуваним повідомленням. Приймач повідомлень чітко налаштований на виконання підходу розпізнання "не очікуване" та "очікуване" повідомлення. У випадку звичайного пошуку повідомлення N повідомлень, які знаходяться перед ним, спочатку повинні копіюватися з буфера сокета у додаткову пам'ять, яка називається "пулом прийому неочікуваних повідомлень". У випадку прямого пошуку повідомлення очікування, реалізованому в даній роботі, всі пропущені неочікувані повідомлення розпізнаються і обробляються за допомогою розробленого системного виклику seek_recv().

Продуктивність такої системи в цілому може бути оцінена за допомогою суми часових інтервалів: системного часу та часу користувача, який може бути відстежений в системі приймача повідомлення. Відсоток активного часу процесора, який повинен бути зменшеним у випадку реалізації прямого пошуку очікуваного повідомлення, буде тим метричним показником, який і буде використовуватися для оцінки ефективності реалізованого в роботі механізму. Кожен тест повинен складається із ~80 повторень, кожне із яких включає в себе сукупність операцій відкриття сокета, пересилання 800-1000 пакетів (повідомлень) на нього відповідно до описаної вище політики прийому і закриття відкритого сокета. Система повинна використовувати тільки один активний сокет в один і той же проміжок робочого часу.

На рис. 1 показано активне зменшення часу процесора, яке досягається в ході використання сокетного інтерфейсу прямого пошуку. Різні криві відносяться, відповідно, до N значень неочікуваних повідомлень, які знаходяться перед очікуваним, в діапазоні від 2 до 16. По осі X на графіку в логарифмічному масштабі умовно відображено розмір повідомлення в байтах. По осі Y подаєть-

ся зменшення часу CPU у відсотках для випадку використання сокетів прямого пошуку повідомлень. Слід додати, що кожна точка даних була перевірена що найменше 10 разів. Подані криві виявляють усереднені значення зі стандартними смугами помилок, з метою наочно продемонструвати відхилення експериментальних значень в прооблених тестових дослідженнях. Як можна побачити з графіка залежностей, значення, отримані нижче рівня 0%, вказують на ті ситуації, коли новий інтерфейс все таки знижує продуктивність системи. Особливо це стосується повідомлень малого розміру, для яких смуги розбіжностей результатів тестування є значно більшими. З наведених результатів тестового експерименту на рис. 1 спостерігається, що смуги помилок значно зменшуються для повідомлень з більшими розмірами. Це можна пояснити тим, що тести протікають довший час і стають менш залежними від випадкової поведінки операційної системи та кешування.

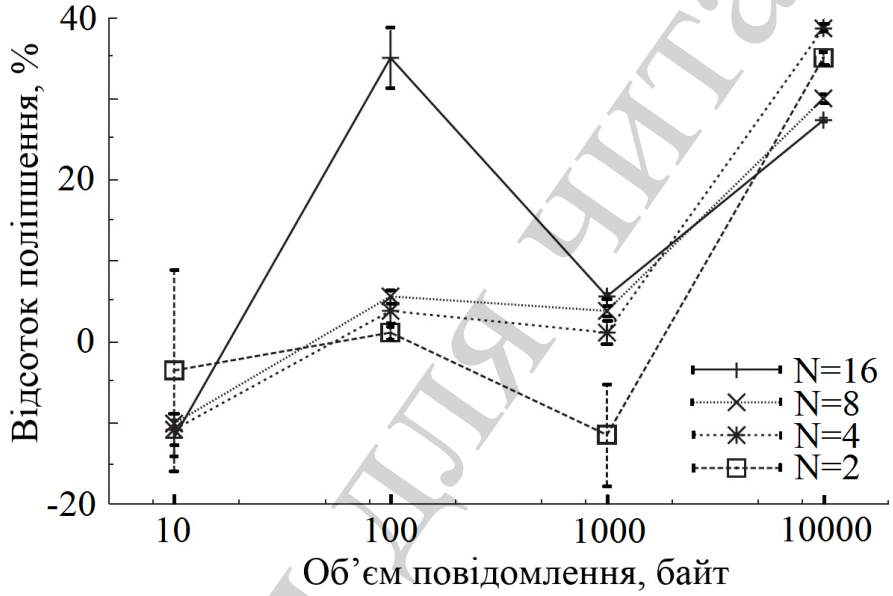

Рис. 1. Вплив різних розмірів повідомлень та розупорядкованої їх кількості N на продуктивність сокетів з прямим пошуком повідомлень

В загальному, отримані на рис. 1 криві виявляють покращення показників продуктивності, які є залежними від збільшення розмірів повідомлень або більших значень N для неочікуваних повідомлень. Ці результати не відображають несподіванки, якщо враховувати необхідність більшої кількості операцій копіювання і додаткових затрат на них для таких же тестів. Для випадку механізму прямого пошуку повідомлень і залученні розробленого інтерфейсу [16] в роботу результати свідчать про помітне зниження процесорного часу виконання на 36–40 % для випадку.

7. Обговорення результатів дослідження механізму прямого пошуку повідомлень

В загальному, виходячи з отриманих результатів досліджень механізму прямого пошуку повідомлень на базі TCP-протоколів в процесі їх обміну на со-

кеті в межах одного з'єднання, спостерігається падіння часу обробки CPU для отриманих очікуваних повідомлень. Це говорить про загальне підвищення продуктивності процесу обміну для випадку, коли очікувані повідомлення повинні бути доставлені в простір користувача. Особливо відчутне зростання продуктивності спостерігається в ході отримання очікуваних повідомлень, які знаходяться поза встановленим порядком їх вибірки. Продуктивність також суттєво зростає у випадку надсилання повідомлень більшого об'єму байт або у випадку надсилання менших за об'ємом повідомлень очікування в більшій їх кількості.

Однак поряд з усіма позитивними результатами слід зауважити (рис. 1), що реалізація процедури пошуку іноді приводить і до загального зниження продуктивності в зв'язку зі збільшенням накладних витрат за рахунок додатково затратної обробки сокетних буферів. Таке зниження спостерігається особливо для малих значень N попередніх неочікуваних повідомлень, а також для невеликих за об'ємом повідомлень в їх малій кількості надсилання. Це має місце, наприклад, в таких проявах, як здійснення управління над списком "дірок" для уже вибраних повідомлень з метою розширення сокетного буфера.

Спостерігаються також і деякі зниження продуктивності системи, що виникають для повідомлень об'ємом 10 байт, які являють собою, наприклад, дещо нереалістичний об'єм представлення даних у добре налаштованих додатках, призначених для вископродуктивних обчислювальних систем. Подібне наближення може бути корисне для дослідження тестування додатків, для яких важко координувати процес комунікації, або за допомогою методу набору мікробенчмарк-міток виявляти і порівнювати базовий час затримки в каналі зв'язку.

В процесі тестування виявлено ще один цікавий результат – це падіння продуктивності в наближенні розміру повідомлень до об'єму 1000 байт. Зазначений розмір для повідомлення є найближчим до розміру корисного завантаження TCP-пакета, тобто 1460 байт. Отже, відносні витрати на керування списком дірок від зчитаних повідомлень у цьому випадку є дещо більшими, ніж у інших наведених випадках.

Результати дослідження демонструють, що інтерфейс сокетів, які здійснюють прямий пошук очікуваного повідомлення, може бути корисний для комунікації та передачі даних на базі TCP-протоколів. Однак за отриманими результатами швидкісна потужність бібліотеки для передачі повідомлень навколо цього інтерфейсу все-таки в повному масштабі не реалізується. Досвід роботи з кодом набору мікробенчмарк-міток підказує, що потрібно здійснити лише деякі незначні зміни в окремих функціях для використання сокетів прямого пошуку повідомлень. Інтегруючи даний інтерфейс в бібліотеку MPI можна значно поліпшити додаткові уявлення про зручність його використання в наступному. Наприклад, прикладним додаткам спочатку потрібно буде перевірити заголовки повідомлень за допомогою звичайних мережевих сервісів, можливо підглянути чи підхопити їх, перш ніж отримати інформацію про їх кількість, необхідну для здійснення процедури пошуку.

З іншої сторони фактична продуктивність, швидше за все, буде змінюватися в залежності від кількості надісланих повідомлень, відповідно до якої неупорядковано отримуються повідомлення в черзі. Як відомо, результат набору мік-

робенчмарк-міток виявляє тільки результат величини процесорного часу обробки, що береться до розгляду, а не його поєднання з часом затримки в ході комунікації. Відповідно до методології [22] ці результати повинні би бути скомпоновані для опису загальної продуктивності запропонованої системи. Очікувані результати також повинні бути в значній мірі незалежними від фізичних проміжних пристроїв використання чи кількості сокетів. Оскільки зміни в операційній системі обмежуються саме сокетним рівнем, то вони строго враховують базовість кожного сокета.

Сокети з інтерфейсом прямого пошуку повідомлень можуть призвести також до поліпшення продуктивності поза межами домена кластерної обробки даних. Наприклад, для більш ефективної роботи TCP сучасні HTTP-реалізації передають кілька одночасних запитів та відповідей на встановленому однонапрямленому конвеєрному з'єднанні [23]. Виходячи із загальних міркувань, сокети, які здійснюють прямий пошук повідомлень, можуть бути задіяні і для паралельної обробки даних через встановлений однонапрямлений зв'язок. Це стосується різнорідних потокових зчитувань, поділу інформації тобто її фрагментації, а також відображення вмісту інформації в різних частинах з'єднання.

Як видно з аналізу дослідження механізму прямого пошуку повідомлень, необхідно враховувати і кількість очікуваних повідомлень, які знаходяться в черзі вибірки в межах одного з'єднання. З експериментальних результатів, отриманих тестуванням реалізованої системи доставки методом набору мікробенчмарк-міток видно, що дана система добре розрізняє очікувані та неочікувані повідомлення зі змінним об'ємом байт за рахунок розширеного сокетного інтерфейсу. Однак залежність продуктивності такої системи для повідомлень великого об'єму байт буде в великій мірі залежати від механізму фрагментації їх на відповідні частини перед надсиланням.

В майбутньому планується провести дослідження використання механізму прямого пошуку та розширеного сокетного інтерфейсу, інтегрованого в повнофункціональну діючу бібліотеку, відповідальну за обмін повідомленнями чи бібліотеку комунікації. Важливим в цьому дослідженні також буде братися до уваги залежність продуктивності (швидкодії) системи обміну з розширеним інтерфейсом від різноманітної фрагментації повідомлень.

8. Висновки

1. В результаті проведеного дослідження практично реалізовано механізм прямого пошуку хаотично розміщених в стеку очікуваних повідомлень в ході встановленої TCP-комунікації між робочими машинами. Для отримання результатів дослідження система з прямим пошуком повідомлень була протестована методом простого мікробенчмаркінгу. Кожен тест містив ~80 повторень, кожне з яких включало пересилання 800-1000 пакетів та інші неохідні операції.

2. В реалізації механізму використано розширений сокетний TCP-інтерфейс, розроблений на базі традиційного, який відслідковував і реалізував отримання повідомлення з будь-якого стекового положення в межах одного з'єднання, обминаючи затратне копіювання.

3. В процесі відшукання хаотично розміщених очікуваних повідомлень в черзі було використано процедуру багаторазового їх пошуку. Поряд з отриманням повідомлень звільняються їх буфери для розширення пам'яті загального сокетного буфера.

4. За отриманими результатами тестування спостерігається скорочення часу обробки CPU для отриманих повідомлень на 36–40 %. Загальне зростання продуктивності процесу обміну має суттєвий прояв у випадку надсилання повідомлень більшого об'єму або великої кількості малих повідомлень. Однак при наближенні об'єму повідомлень до характерного розміру корисного завантаження TCP-пакета, спостерігається зниження продуктивності процесу обміну.

Література

1. Distributed processing of very large datasets with DataCutter / Beynon M. D., Kurc T., Catalyurek U., Chang C., Sussman A., Saltz J. // Parallel Computing. 2001. Vol. 27, Issue 11. P. 1457–1578. doi: https://doi.org/10.1016/s0167-8191(01)00099-0
2. Small-file access in parallel file systems / Carns P., Lang S., Ross R., Vilayannur M., Kunkel J., Ludwig T. // 2009 IEEE International Symposium on Parallel & Distributed Processing. 2009. doi: https://doi.org/10.1109/ipdps.2009.5161029
3. Managing Big Data with Information Flow Control / Pasquier T. F. J.-M., Singh J., Bacon J., Hermant O. 2015. URL: http://tfjmp.org/files/publications/cloud-2015.pdf
4. Мельник В. М., Багнюк Н. В., Мельник К. В. Вплив високопродуктивних сокетів на інтенсивність обробки даних // ScienceRise. 2015. Т. 6, № 2 (11). С. 38–48. doi: https://doi.org/10.15587/2313-8416.2015.44380
5. Infiniband Trade Association. URL: http://www.infinibandta.org
6. Netgauge: A Network Performance Measurement Framework / Hoefler T., Mehlan T., Lumsdaine A., Rehm W. // High Performance Computing and Communications. 2007. P. 659–671. doi: https://doi.org/10.1007/978-3-540-75444-2_62
7. Majumder S., Rixner S., Pai V. S. An Event-driven Architecture for MPI Libraries // In Proceedings of the 2010 Los Alamos Computer Science Institute Symposium. 2010. URL: https://vjpai.github.io/Publications/majumder-lacsi04.pdf
8. Gilfeather P., Maccab A. B. An Extensible Message-Oriented Offload Model for High-Performance Applications // Los Alamos Computer Science Institute SC R71700H29200001. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.217.1085&rep=rep1&type=pdf
9. Veeraraghavan M., Jukan A. A hybrid networking architecture // University of Virginia. 2010. URL: https://pdfs.semanticscholar.org/62bb/a7fcb0e97adbf623a569c160d20843022b08.pdf
10. Aydin S., Bay O. F. Building a high performance computing clusters to use in computing course applications // Procedia – Social and Behavioral Sciences. 2009. Vol. 1, Issue 1. P. 2396–2401. doi: https://doi.org/10.1016/j.sbspro.2009.01.420
11. Pratt I., Fraser K. Arsenic: a user-accessible gigabit Ethernet interface // Proceedings IEEE INFOCOM 2001. Conference on Computer Communications.

Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213). 2001. doi: https://doi.org/10.1109/infcom.2001.916688

12. Fenech K. Low-latency inter-thread communication over Gigabit Ethernet. University of Malta, 2005. 180 p. URL: http://citeseerx.ist.psu.edu/viewdoc/download?DOI=10.1.1.192.2018&rep=rep1&type=pdf

13. Ethernet Networking Interface // Techopedia. 2019. URL: https://www.techopedia.com/definition/24959/ethernet-networking-interface

14. Stream Control Transmission Protocol / Stewart R., Xie Q., Morneault K., Sharp C., Schwarzbauer H., Taylor T. et. al. // 2000. doi: https://doi.org/10.17487/rfc2960

15. Sockets API Extensions for Stream Control Transmission Protocol (SCTP) / Stewart R., Xie Q., Yarroll L., Wood J., Poon K., Tuexen M. // IETF Internet Draft. 2005. URL: https://tools.ietf.org/pdf/draft-ietf-tsvwg-sctpsocket-06.pdf

16. Мельник В.М. Моделювання механізму пошуку повідомлень в процесі їх обміну на базі протоколів TCP // Науковий журнал "Комп'ютерно-інтегровані технології: освіта, наука, виробництво". 2017. № 28-29. С. 20–24.

17. Liyanage M., Ylianttila M., Gurtov A. Fast Transmission Mechanism for Secure VPLS Architectures // 2017 IEEE International Conference on Computer and Information Technology (CIT). 2017. doi: https://doi.org/10.1109/cit.2017.46

18. Performance analysis of asynchronous Jacobi's method implemented in MPI, SHMEM and OpenMP / Bethune I., Bull J. M., Dingle N. J., Higham N. J. // The International Journal of High Performance Computing Applications. 2014. Vol. 28, Issue 1. P. 97–111. doi: https://doi.org/10.1177/1094342013493123

19. Implementation of the simplified communication mechanism in the cloud of high performance computations / Melnyk V., Bahnyuk N., Melnyk K., Zhyharevych O., Panasyuk N. // Eastern-European Journal of Enterprise Technologies. 2017. Vol. 2, Issue 2 (86). P. 24–32. doi: https://doi.org/10.15587/1729-4061.2017.98896

20. Обзор некоторых пакетов измерения производительности кластерных систем. URL: https://www.ixbt.com/cpu/cluster-benchtheory.shtml

21. Computer Network & TCP Congestion Control. URL: https://www.geeksforgeeks.org/computer-network-tcp-congestion-control/

22. 7 Steps Needed for Successful Benchmarking, using the COMPARE Method. URL: https://www.compare2compete.com/en/blog/7-steps-needed-for-successful-benchmarking-using-the-compare-method/

23. HTTP Definition. URL: https://techterms.com/definition/http